Mitteilung Nr. 231

High-Level Design in der Lehre: Entwurf eines Taschenrechner-ICs

Norman Hendrich, Andreas Mäder

FBI-HH-M- 231-93

November 1993

Fachbereich Informatik/TECH
Universität Hamburg
Troplowitzstr. 7
22529 Hamburg

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Das High-Level Entwurfsprojekt `18.332`

Dieser Bericht beschreibt den Versuch, High-Level Design in einem Entwurfsprojekt einzusetzen (Vorl.Nr. `18.332` im WS 92/93). In einem Semester sollten die Studenten also zunächst eine algorithmische Spezifikation erstellen und dann mittels High-Level Synthese und kommerzieller VLSI-Entwurfssoftware daraus einen Chip entwerfen.

Im ersten Teil dieses Berichts beschreiben wir kurz unsere Ziele und die Erfahrungen mit dem Projekt. Dabei versuchen wir, die gestellte Aufgabe — Entwurf eines Taschenrechner ICs — und die Motivation für die Auswahl der verwendeten Sprachen und Werkzeuge — MIMOLA für die High-Level Synthese und SOLO 1400 für die Layouterstellung — zu erläutern.

Der zweite Teil des Berichts enthält dann eine detaillierte Spezifikation und technische Dokumentation für den im Rahmen des Projekts entworfenen Taschenrechner. Dabei werden die beiden alternativen Entwürfe `bincalc` und `bcdcalc` vorgestellt und bewertet. Der `bcdcalc`-Chip wird über EUROCHIP gefertigt.

1.1 Die Ziele

Die Ausbildung im Bereich „VLSI“ besteht traditionell aus Vorlesungen über die technologischen Grundlagen (Bipolar, NMOS, CMOS, etc.) und Algorithmen für VLSI, insbesondere Simulationsalgorithmen, Plazierung & Verdrahtung und Logiksynthese. Dazu kommen Praktika, in denen die Studenten kleinere Full-Custom und Standardzellentwürfe erstellen. Dabei kommen bei uns MAGIC und CADENCE/EDGE für die Full-Custom Versuche und SOLO 1400 für die Standardzellentwürfe zum Einsatz.

In den Entwurfsprojekten sollen Studenten, die bereits die Vorlesungen gehört und das Praktikum absolviert haben, dann weitgehend selbständig einen wirklichen VLSI Entwurf durchführen. Auch für diese Projekte setzen wir hauptsächlich SOLO 1400 ein, weil die Bedienung einfacher zu erlernen ist als bei CADENCE/SOLO 2030. Die Beschreibung der Entwürfe mit der hierarchischen Sprache MODEL des SOLO 1400 Systems ermöglicht außerdem den Verzicht auf das zeitraubende Zeichnen von Schematics.

In früheren Entwurfsprojekten wurden mehrfach Entwürfe als Aufgabe gestellt, die durch Kooperationen mit lokalen industriellen Partnern motiviert waren. Dabei kristallisierte sich eine Art „Standard“-Entwurfsablauf heraus: Ausgehend von einer allgemeinen Spezifikation wurde zunächst ein C- oder Pascalprogramm erstellt, um die benötigten Algorithmen zu simulieren. Anschließend wurde von den Projektteilnehmern ad hoc eine Hardwarestruktur ausgewählt, die Algorithmen an diese Hardware angepaßt, neue C- oder Pascalprogramme (zum Modellierung der RT-Ebene!) geschrieben und neu simuliert. Dann folgte die

Erstellung einer MODEL Beschreibung für diese Hardware und die Layouterstellung und Simulation mit SOLO 1400.

Bei diesem Entwurfsablauf ist der Einsatz von High-Level Werkzeugen sowohl für die Spezifikation/Simulation als auch für Syntheseschritte offenbar geboten. Die bisher benutzten stand-alone Programme in Pascal oder C zur Simulation werden dazu durch entsprechende Programme in der Eingabesprache der High-Level Werkzeuge ersetzt. Danach können die vorher manuell ausgeführten Transformationen automatisch von den Syntheseprogrammen vorgenommen werden, und außerdem werden die Simulationen auf den verschiedenen Entwurfsebenen drastisch erleichtert.

Damit ergaben sich für das Projekt 18.332 also folgende Ziele:

- Das Projekt sollte versuchen, die Top-Down Entwurfsphilosophie an einem größeren Beispiel möglichst weit zu verwirklichen. Der Hauptteil des Projekts war dementsprechend für die Erstellung und Bewertung der Spezifikation auf Algorithmenebene reserviert.

  Zusätzlich sollte versucht werden, mögliche Bottom-Up Effekte vorauszusehen und die algorithmische Spezifikation schon darauf auszurichten.

- Der Einsatz eines High-Level Synthesesystems sollte es erlauben, eine größere Anzahl von Entwurfsalternativen auszuprobieren und damit zu einer möglichst guten Lösung zu gelangen.

  Daher sollte die Entwurfsaufgabe möglichst die parallele Entwicklung mehrerer Alternativen durch einzelne Studenten/Gruppen erlauben. Nur die beste dieser Alternativen sollte dann wirklich bis zu einem Layout gebracht werden.

- Nur wenige High-Level Synthesesysteme haben einen Zustand erreicht, in dem sie auch von „Nicht-Experten“ bedient werden können.

  Das Entwurfsprojekt sollte zusätzlich zeigen, daß die MIMOLA und MIM2SOLO Werkzeuge auch von Studenten effektiv eingesetzt werden können.

- Als Nebeneffekt ergab sich außerdem die Möglichkeit, die selbstentwickelten Werkzeuge MIM2SOLO an weiteren Designbeispielen zu testen.

1.2 Die Aufgabe: Ein Taschenrechner

Unter den Rahmenbedingungen des Projekts erwies sich die Auswahl einer geeigneten Entwurfsaufgabe als eigenes Problem. Die Aufgabe sollte natürlich interessant genug sein, um die Studenten zu motivieren. Auf der anderen Seite mußte die Aufgabe einfach genug sein, um innerhalb eines Semesters bearbeitet werden zu können.

Weitere Beschränkungen ergaben sich aus den verfügbaren Entwurfswerkzeugen. Für die Entwurfsschritte ab RT-Ebene, also für Netzlistengenerierung, Simulation und Layouterstellung bot sich das SOLO 1400-System an, mit dem wir bereits sehr gute Erfahrungen in der Lehre gesammelt hatten.

Für die High-Level Synthese standen uns zum Zeitpunkt des Projekts MIMOLA [Marwedel 85, Jöhnk & Marwedel 89] und das Olympus System [Ku & Micheli 90] aus Stanford zur Verfügung. Während MIMOLA auf die Synthese von Prozessorstrukturen mit einem zentralen Kontrollfluß ausgerichtet ist, erlaubt Olympus auch die Beschreibung mehrerer paralleler, kommunizierender Prozesse.

Aufgrund der guten Erfahrungen und dem durchgängigen Design-Flow mit MIM2SOLO [Hendrich, Lohse & Rauscher 92a] entschieden wir uns für den Einsatz des MIMOLA Systems. Ein großer Vorteil bei der Verwendung von MIMOLA ist, daß die Studenten ihre Entwürfe zunächst in Pascal programmieren und austesten können, da die Spezifikaton von Algorithmen in MIMOLA bis auf unbedeutende Ausnahmen der Syntax und Semantik von Pascal entspricht. Außerdem unterstützt MIM2SOLO neben Gatternetzlisten auch die Verwendung der SOLO 1400 Makrozellgeneratoren für RAMs und ROMs. Allerdings müßte sich die Entwurfsaufgabe dann mit einem CISC-artigen Prozessor lösen lassen, um eine effiziente Synthese mit MIMOLA zu erlauben.

Schließlich wählten wir als Aufgabe eine Taschenrechner-CPU, allerdings mitsamt der dazugehörigen Peripherie. Diese Aufgabe erschien sowohl für eine MIMOLA Synthese wie auch aus didaktischen Gesichtspunkten als geeignet:

- Die Aufgabe ist intuitiv und leicht zu verstehen. Der Funktionsumfang kann jederzeit an den Stand des Projekts angepaßt werden (Erweitern um/Weglassen von Funktionen)
- Es gibt eine große Auswahl von möglichen Algorithmen und Architekturen (seriell/parallel, Auswahl der Arithmetik)
- Die Performanzanforderungen sind gering. Daher kommt auch eine von Anfängern geschriebene und daher nicht unbedingt optimale Spezifikation für den Entwurf in Frage
- Die Aufgabe erfordert auch die Spezifikation der „Umgebung" des Chips (Ansteuerung einer Anzeige, Tastenkodierung und Entprellung)
- Ein Taschenrechner kann auf jeden Fall als Prozessor (mit einem einzelnen Kontrollfluß) realisert werden — mit möglicherweise recht komplexem Mikroprogramm. Damit ist MIMOLA sehr gut für die Synthese geeignet.

Um die Machbarkeit zu beweisen, erstellten wir noch kurz vor Beginn des Projekts eine MIMOLA Spezifikation für einen minimalen Taschenrechner mit BCD Arithmetik. Dieser

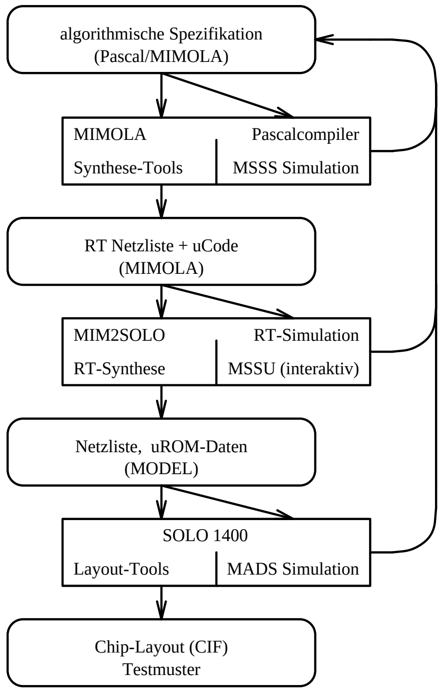


Abbildung 1: MIM2SOLO Design-Flow

erste Entwurf umfaßte lediglich serielle Addition und Multiplikation für Integerzahlen mit 8 Dezimalstellen und eine Ansteuerung für eine Sieben-Segment Anzeige. Obwohl der Entwurf noch viele Fehler enthielt, konnten wir damit den Studenten auch ein erstes, größeres Beispiel für eine synthetisierbare MIMOLA Spezifikation präsentieren.

1.3 Der Design-Flow

Der konsequente Einsatz einer Top-Down Entwurfsmethodik von der algorithmischen Spezifikation bis hinunter zum Chip-Layout war, wie oben beschrieben, eines der erklärten Ziele des Entwurfsprojektes.

Dies läßt sich mit dem MIM2SOLO System auch realisieren, der ideale Design-Flow ist in Abbildung 1 skizziert. Ausgehend von einer natürlichsprachlichen Formulierung der Aufgabenstellung wird zunächst eine algorithmische Spezifikation erstellt und ausgetestet. Diese algorithmische Spezifikation kann von Anfang an in MIMOLA geschrieben und simuliert werden — dazu erzeugt der Simulator MSSS des MIMOLA Systems aus der MIMOLA Beschreibung ein Pascalprogramm und führt dieses aus.

Alternativ dazu kann auch ein Pascalprogramm stehen, das mit einer normalen Pascal Entwicklungsumgebung compiliert und getestet wird. Dieser Weg war für das Projekt günstig, weil viele der Studenten zu Hause über einen PC mit Pascal-Compiler verfügen: Dadurch konnten Sie zeitlich und räumlich unabhängig von unseren Workstations ihre Programme entwickeln.

Anschließend wird für die High-Level Synthese die algorithmische Spezifikation um eine Bibliothek der für die Hardwaresynthese zulässigen Module erweitert und die Semantik von Prozeduraufrufen definiert. Das von MIMOLA verwendete List-Scheduling (mit der Option zur Parallelisierung á la ASAP) ermöglicht auch für sehr umfangreiche Algorithmen eine schnelle Synthese.

Die von MIMOLA verwendeten Heuristiken für Allocation und Binding erlauben die Steuerung der Synthese durch die Angabe von Kosten für die zur Verfügung stehenden Operatoren aus der Modulbibliothek. So kann zum Beispiel zwischen Ripple-Carry oder Carry-Look-Ahead Addierern ausgewählt werden. Die nach dem Binding benötigten Module werden dann über Multiplexer verbunden. Schließlich erzeugt der retargierbare Compiler MSSB ein horizontales Mikroprogramm für die generierte Hardwarestruktur. Die Hardwaresynthese erfolgt dabei sehr schnell. Ein kompletter Durchlauf von der Design-Analyse bis zur Mikroprogrammgenerierung für `bcdcalc` etwa erfordert weniger als fünf Minuten CPU-Zeit (SparcStation 10/30).

An dieser Stelle kann die erzeugte RT-Netzliste mit zugehörigem Mikroprogramm mit dem MIMOLA Struktursimulator MSSU interaktiv simuliert werden. Durch die vielfältigen Möglichkeiten zum Setzen von Breakpoints und zur Beobachtung und Modifikation von Registerinhalten gestaltet sich eine evtl. notwendige Fehlersuche sehr einfach.

Erst wenn die RT-Simulationen zufriedenstellend verlaufen, werden mit MIM2SOLO aus der RT-Netzliste eine Logiknetzliste und aus dem Mikroprogramm eine ROM-Beschreibung als Eingabe für SOLO 1400 erzeugt. Die Umsetzung der von MIMOLA verwendeten generischen Operatoren und Multiplexer (zum Beispiel n Bit Addierer) in Beschreibungen auf Gatterebene benutzt dabei eine ebenfalls generische Bibliothek RTLIB von RT-Konstrukten.

Die von MIM2SOLO erzeugte MODEL Beschreibung kann zwar direkt für die Layoutwerkzeuge und den Gatterebenen-Simulator von SOLO 1400 verwendet werden. Manchmal ist es jedoch wünschenswert, noch einige Modifikationen an der MODEL Beschreibung vorzunehmen. Durch die Verwendung der Module aus der RTLIB ist die MODEL Beschreibung sehr übersichtlich und einfach editierbar — anders als etwa die vom Olympus-System erzeugten Gatternetzlisten.

Ansatzpunkte für die Modifikation der MODEL Beschreibung sind zum Beispiel zusätzliche Logik für Erhöhung der Testbarkeit — sofern dies nicht schon in der MIMOLA Spezifikation möglich war — und der Einsatz von handoptimierten Modulen, etwa für ALUs mit mehreren Funktionen. Ein eigenes Programm übernimmt das wegen Bugs in den SOLO 1400 Programmen MODEL und MADS notwendige Aufspalten von Bussen in Teilbusse mit jeweils weniger als 32 Bit Breite.

Für die detaillierte Beschreibung des Entwurfsablaufs mit SOLO 1400, ausgehend von der MODEL Beschreibung, sei auf die Dokumentation [ES2 90], [ES2 91] verwiesen. Als Ergebnis erzeugt SOLO 1400 ein Chip-Layout im CIF Format und die dazugehörigen Testmuster.

2 Erfahrungen mit High-Level Design

In diesem Abschnitt wollen wir die Erfahrungen, die wir im Laufe dieses Projekts gesammelt haben, zusammenfassen und eine Bewertung der eingesetzten Werkzeuge vornehmen. Bei der Beschreibung der beiden Entwürfe `bincalc` und `bcdcalc` in den Kapiteln 3 und 4 werden, entsprechend der zeitlichen Entwicklung während des Projekts, viele der Punkte noch genauer erläutert.

2.1 Das gewählte Entwurfsvorgehen

2.1.1 Die Aufgabenstellung

Das Beispiel des Taschenrechners hat sich im Laufe des Projekts sehr gut bewährt, da es unseren Anforderungen, denen der „Veranstalter“ solch eines Projekts, genau entsprach.

Dies galt zum einen in Bezug auf die Studenten, für die die Aufgabe intuitiv verständlich war, was dazu führte, daß ohne größeren Einarbeitungsaufwand direkt mit der Arbeit begonnen werden konnte. Außerdem waren die Studenten von Anfang an sehr gut motiviert und haben, da wir entsprechende Freiheitsgrade gelassen hatten, viele eigene Ideen realisieren und bewerten können.

In Bezug auf die inhaltlichen Ziele der Lehre, also den zu vermittelnden Stoff oder besser die zu vermittelnden Fähigkeiten, bot der Entwurf eines Taschenrechners folgende Vorteile:

- Die Aufgabenstellung konnte „vage“ natürlichsprachlich formuliert werden, was den üblichen „Spezifikationen“ entspricht, die die Studenten später im industriellen Umfeld erhalten werden.
- Bei einer Umsetzung dieser Spezifikation gab es dementsprechend sehr viele Freiheitsgrade, sowohl in Bezug auf die zu verwendenden Algorithmen, als auch in Bezug auf die zu realisierende Architektur. So konnte eine größere Anzahl alternativer Entwürfe erstellt werden.
- Da die Aufgabe vom Umfang keinen festen Rahmen besitzt, konnte die Spezifikation, entsprechend dem zeitlichen Ablauf des Projekts, um zusätzliche Funktionen erweitert werden.
- Nach der Fertigung der Schaltung wird es möglich sein, den Chip mit einem Minimum an externem Schaltungsaufwand in Betrieb zu nehmen.

  An dieser Stelle soll noch einmal betont werden, wie wichtig es für die Motivation der Studenten ist, daß IC-Designs nicht nur bis hin zum abgabefertigen Layout entworfen werden, sondern daß sie auch „wirklich“ gefertigt werden.
- Er ließ sich vergleichsweise gut mit den gewählten Entwurfsprogrammen realisieren.

2.1.2 Top-Down Design

Während der letzten Jahre stellten neue Anwendungsgebiete, wie digitale Signalverarbeitung, Kommunikationstechnik, Bildverarbeitung und viele andere, Anforderungen an Größe, Zuverlässigkeit und Datendurchsatz, die von herkömmlichen Softwarelösungen auf Standardprozessoren nicht mehr erfüllt werden können. Um die Entwürfe bei zunehmender Komplexität — Stichwort: systems in silicon — und immer kürzer werdenden Designzyklen noch handhaben zu können, werden die Designs auf immer höheren Abstraktionsebenen beschrieben, wobei die Umsetzung dieser Beschreibungen auf das Layout auf Maskenebene weitgehend automatisch erfolgt:

- Standardzell-Techniken, die Benutzung von Makrozellgeneratoren und automatische Tools für Plazierung & Verdrahtung erlauben eine schnelle Umsetzung von Logik (und RT-Ebene) auf das Maskenlayout.
- Durch die Logiksynthese kann eine funktionale Abstraktion vorgenommen werden. Die Logiksynthese ist dabei „gebräuchlicher“ Bestandteil von Entwurfsumgebungen wie SOLO 1400 und CADENCE/SOLO 2030.
- Durch Verhaltensbeschreibungen, auch komplexer Prozesse, und den Einsatz von High-Level Synthese können dann auch höhere Abstraktionsebenen im Entwurfsprozeß gehandhabt werden. Die Vorteile einer solchen Vorgehensweise liegen in:
  - verhaltens-/prozeßorientierter Spezifikation
  - früher Einsatz von Simulationswerkzeugen sowie deren durchgängige Benutzung auf verschiedenen Abstraktionsebenen
  - Handhabbarkeit komplexerer Entwürfe

Mit dem Entwurfsprojekt `18.332` haben wir versucht, dieser Entwicklung auch in der Lehre Rechnung zu tragen.

Der Entwurfsprozeß stellt sich damit — im Idealfall — als ein „Top-Down“ Durchlauf durch die verschiedenen Hierarchieebenen dar. Leider ist diese Vorstellung stark idealisiert, denn in der Praxis zeigt sich, daß der „Anwender“, der seine Schaltung auf einer hohen Abstraktionsebene spezifiziert und dann mit Hilfe der oben genannten Automatismen umsetzen läßt, zwar einen Chip erhält, der die spezifizierte Funktion erfüllt, der aber oft weder „effizient“ noch praktisch einsetzbar ist. So lassen sich qualitative Eigenschaften der Schaltung, wie Größe des ICs, Arbeitsgeschwindigkeit und Leistungsaufnahme nicht in der Verhaltensbeschreibung spezifizieren und können auch bei der Steuerung der Synthese nur eingeschränkt — meist als Tradeoff zwischen Größe und Geschwindigkeit — beeinflußt werden.

Bei dem Entwurf wirken, im Gegensatz zu einem reinen „Top-Down“ Vorgehen „Bottom-Up“ Effekte — Einflüsse aus den unteren Abstraktionsebenen. Sie beeinflussen sowohl die Spezifikation, als auch die Vorgehensweise auf den höheren Abstraktionsebenen des Entwurfs. Diese Effekte können dabei qualitativ in zwei Gruppen unterteilt werden:

Technologie: Elektrotechnische Grundlagen und die schaltungstechnische Realisierung von Gattern beeinflussen dabei beispielsweise:

– die maximale Taktfrequenz und die Leistungsaufnahme der Schaltung — Rückwirkung auf die Spezifikation
– die Benutzung bestimmeter Schaltungstechniken, wie z.B. precharging — Erweiterung der Architektur
– die Formulierung logischer Ausdrücke, z.B. die Benutzung von Komplexgattern in MOS-Technik — Anpassung der logisch funktionalen Beschreibung

Algorithmen: Der Designer sollte die in den Entwurfswerkzeugen verwendeten Algorithmen kennen, um wirklich „gute" Entwürfe machen zu können. Diese Aussage scheint zwar übertrieben, trifft aber noch für fast alle Syntheseschritte zu:

– Die Qualität von Plazierung & Verdrahtung hängt bei einigen Werkzeugen sehr stark von Eigenschaften der Netzliste ab, manchmal sogar von der Reihenfolge von Blöcken und Signalen (z.B. in SOLO 1400).
– Bei der Logiksynthese kann die Art und Weise, wie (logisch äquivalente) Ausdrücke formuliert sind, entscheidend das Syntheseergebnis beeinflussen.
– Im Bereich der High-Level Synthese tritt dieser Effekt noch extremer zutage. Minimale Änderungen in der Eingabebeschreibung können zu extrem unterschiedlichen Ergebnissen führen, eine Lokalität ist oft nicht gegeben. Deshalb muß der Entwerfer schon über eine gewisse Erfahrung beim Umgang mit einem System verfügen, um eine „synthesegerechte" Eingabe zu erstellen.

Der Grund für dieses Verhalten liegt in der Tatsache, daß alle oben ganannten Problemstellungen NP-vollständig sind und nur über bestimmte Heuristiken angegangen werden können.

Letztendlich kann der Entwurfsprozeß als ein „oszillierender Top-Down" Ablauf angesehen werden, der mit einer mehr oder minder abstrakten Spezifikation beginnt und mit dem fertigen Layout und einem Satz von Testmustern endet. Iterationen innerhalb dieses Ablaufs lassen ihn die Hierarchieebenen wie ein Jo-Jo durchlaufen: ausgehend von einem obersten Punkt, durch die Hierarchie hinablaufend bis zu einem Umkehrpunkt, anschließend zurückkehrend zu höheren Abstraktionsebenen und wieder verfeinernd durch die Hierarchie. Die Gründe für diese Iterationen können Fehler bei der „Top-Down" Vorgehensweise sein, Restriktionen, die in den unteren Hierarchieebenen wirksam werden (die oben erwähnten „Bottom-Up" Effekte) oder einfach der Versuch des Entwerfers, verschiedene alternative Realisierungen seiner Spezifikaition zu erzeugen, um eine „beste" auszuwählen.

Ein „guter" Designer wird dabei versuchen (wie ein guter Jo-Jo Spieler), möglichst lange Wege zwischen den Richtungswechseln zurückzulegen, die nur durch die Implementation von Entwurfsalternativen bedingt werden.

Alternativen

Bei dem Entwurfsprojekt konnten entsprechend der Aufgabenstellung (s.o.) viele solcher Entwurfsalternativen verfolgt werden, wobei die Wahl der Arithmetik — BCD oder binär — sich über unterschiedliche Algorithmen, also unterschiedliche Verhaltensbeschreibungen, bis auf die höchste Hierarchieebene auswirkt und in den beiden hier vorgestellten Entwürfen `bincalc` und `bcdcalc` mündet. Auf niedrigeren Abstraktionsebenen sind mögliche Alternativen dann:

- bei dem Aufbau der Verbindungsstrukturen: Busarchitekturen, gemultiplexte Eingänge
- bei der Behandlung von Variablen des Quellprogramms: lokal, global
- bei der Realisierung der Rechenwerke: PLA oder Gatterlogik
- bei der Organisation der Speicher: Größe, Wortbreite, Anzahl

Wie bei jedem größeren Entwurf sind so eine Anzahl Entwurfsentscheidungen gefällt worden, wobei die meisten der oben erwähnten Punkte in den Kapiteln 3 und 4 weiter ausgeführt sind.

Durchgängigkeit der Top-Down Vorgehensweise

Der Ablauf dieses Projekts zeigt sehr gut das oben beschriebene „Jo-Jo"-artige Entwurfsvorgehen. Entsprechend den beiden unterschiedlichen Algorithmen, gab es zwei komplette Top-Down Durchläufe, von der Spezifikation bis hin zum Layout. Innerhalb dieser Durchläufe traten weitere Iterationen jeweils nur innerhalb der beiden Werkzeugumgebungen zu MIMOLA und SOLO 1400 auf, während die Verbindung zwischen diesen beiden durch MIM2SOLO eine Art Grenze gebildet hat. Die Gründe für diese Iterationen werden in dem Abschnitt 2.2 noch detailliert beschrieben. Es sind jeweils typische Bottom-Up Effekte, zum einen die synthesegerechte Eingabe bei MIMOLA und zum anderen Probleme bei der automatischen Plazierung & Verdrahtung bei SOLO 1400.

Testbarkeitsüberlegungen

Ein wichtiger Punkt, den ein Entwerfer immer berücksichtigen muß, ist die Testbarkeit seiner Schaltung. Während es für diesen Bereich auf den unteren Entwurfsebenen viele bewährte Strategien gibt, um die Testbarkeit der Schaltung zu gewährleisten (built-in self-test, scan-path ...), ist deren Umsetzung im High-Level Design nicht einheitlich gewährleistet; hier sind die Konzepte von Werkzeug zu Werkzeug sehr unterschiedlich.

Einige High-Level Synthesesysteme sind in der Lage, automatisch Scanpfade und BIST-Logik in eine RT-Struktur einzufügen und können so, mit Hilfe zusätzlicher Testmustergeneratoren, eine bestimmte Fehlerüberdeckung garantieren. Dies entspricht konsequent der Idee des High-Level Designs, unabhängig von Einflüssen unterer Hierarchieebenen das Verhalten der Schaltung als Eingabe zu spezifizieren. Bei den Abbildungsschritten bis zum Layout werden dann, abhängig von der Technologie, zusätzliche Maßnahmen zur Erhöhung der Testbarkeit (automatisch) getroffen.

Bei der in diesem Projekt verwendeten Tool-Chain — MIMOLA, MIM2SOLO, SOLO 1400— ist dieses Vorgehen leider nicht möglich; deshalb mußten Testbarkeitsüberlegungen explizit im Entwurf berücksichtigt werden.

Um den Entwurf vollständig testen zu können, sind in `bcdcalc` folgende Ergänzungen implementiert worden:

- Speicherselbsttest nach dem Zurücksetzen der Schaltung
- Testmodus-Eingabe über einen Bus
- Scanpfad
- Ausgabe des Mikroprogrammspeichers

Von den oben genannten Architekturerweiterungen sind die ersten beiden schon für die High-Level Synthese beschrieben worden. In diesen Fällen bot sich eine Änderung des Verhaltens der Schaltung durch Erweiterung des MIMOLA-Quellcodes an, da der Selbsttest und die parallele Eingabe auf dieser hohen Ebene (prozedural) beschreibbar waren.

Da der Scanpfad und die ROM-Ausgabe sich nicht in die MIMOLA-Beschreibung einarbeiten ließen, beziehungsweise zu ineffizienten Realisierungen auf der RT-Ebene geführt hätten, wurden sie in der Eingabe für SOLO 1400, der MODEL-Datei, von Hand implementiert.

2.2 Bewertung der eingesetzten Entwurfswerkzeuge

2.2.1 MIMOLA

Im Bereich der High-Level Synthese kann es, wegen der Komplexität der zu bewältigenden Aufgaben und wegen der Größe des zu untersuchenden Suchraums, kein „universelles" Synthesesystem geben. Dementsprechend ist die Eignung eines High-Level Synthesewekzeugs abhängig von der Aufgabenstellung beziehungsweise von der Zielarchitektur des Synthesesystems — beispielsweise ob sich ein Problem eher Datenfluß- oder eher Kontrollfluß-orientiert darstellt.

MIMOLA [Marwedel 85] erzeugt als Zielarchitektur mikroprogrammierte Systeme, mit einem zentralen Steuerwerk und dem aus Funktionseinheiten und Variablenspeicher aufgebauten Operationswerk. Im Operationswerk können Variablen in Registern und/oder in RAMs gespeichert werden; die Datenpfade werden unidirektional über gemultiplexte Eingänge geschaltet. Bei der nachfolgenden Umsetzung mit MIM2SOLO [Hendrich, Lohse & Rauscher 92a, Hendrich, Lohse & Rauscher 92b] können dafür alternativ Multiplexerlösungen oder Tristate-Busse generiert werden.

Entsprechend dieser Architektur ist MIMOLA besonders für Aufgabenstellungen geeignet, die über einen einzigen Kontrollfluß prozedural gesteuert sind, wie Prozessorkerne und ähnliches.

Da bei der Auswahl der Aufgabenstellung im Rahmen dieses Projekts das Werkzeug MIMOLA schon feststand, wurde die Aufgabe „Taschenrechner“ bewußt so gewählt, daß sie für eine Realisierung mit diesem System geeignet schien. Dementsprechend sind unsere Erfahrungen mit dem Einsatz von MIMOLA in diesem Projekt auch sehr positiv. Dazu nachfolgend noch Anmerkungen zu einigen Punkten:

Eingabesprache

Da die Eingabesprache MIMOLA stark Pascal-ähnlich ist, hatten die Studenten während des Projekts keinerlei Probleme die initiale Verhaltensbeschreibung zu erstellen. Vorhandene Programmierkenntnisse konnten direkt umgesetzt werden.

Wie schon im vorangegangenen Abschnitt 2.1.2 beschrieben, kam es allerdings zu „Bottom-Up“ Einflüssen, die eine Änderung der ursprünglichen MIMOLA-Beschreibung notwendig machten. Wie bei allen High-Level Synthesewerkzeugen, muß der Entwerfer prinzipiell wissen, zu welchen Ergebnissen die verwendeten Konstrukte der Verhaltensbeschreibung führen — Kenntnis über die verwendeten Algorithmen und viel Erfahrung, die leider in den seltensten Fällen auf andere Synthesesyteme übertragbar ist. Das Ergebnis dieser Iterationen war dann, ausgehend von einer zwar korrekten aber suboptimalen Lösung, eine „synthesegerechte“ Eingabe für MIMOLA. Im einzelnen kam es zu folgenden Problemen:

Abbildung der Datentypen auf „reale“ Hardware Record-orientierte Datentypen, die sich in Pascal anbieten (Records mit Mantisse und Exponent, jeweils aufgebaut als Arrays von Bits), führen in der Hardwarerealisierung zu sehr komplizierten Adressierungsmethoden. Dementsprechend hat MIMOLA zusätzliche Hardware für die Adreßberechnung (Addierer und Basisadreßregister) einführen müssen.

Bei einigen Datentypen wird sowohl auf gesamte Arrays (Bitstrings) als auch auf Teile davon (z.B. MSB als Vorzeichen) zugegriffen. Durch Maskierungen und Multiplexerlogik wurden diese Mechanismen in der RT-Struktur realisiert.

In beiden Fällen hat eine weniger „Pascal-strukturierte“ Deklaration der Datentypen den nach der Synthese benötigten Hardwareaufwand senken können.

Prozedurale Programmierung Eine Strukturierung des Programms mit Prozeduren hat, analog zur Reduktion der Quellcodes in der Programmierung, eine drastische Verringerung der Anzahl der Mikroprogrammworte zur Folge — was natürlich von Entwerfer gewünscht wird. Entsprechend zur Programmierung heißt das allerdings für die Hardware, daß auch ein Stack zur Verwaltung der Prozeduradressen notwendig wird. Deshalb ist, im Gegensatz zur Programmierung, eine prozedurale Aufteilung des MIMOLA-Programms nur dann sinnvoll, wenn die Prozeduren mehrfach benutzt werden und sich der zusätzliche Hardwareaufwand für die Vewaltung der Prozedurschachtelung lohnt — im Gegensatz zur Programmierung, wo es auf Übersichtlichkeit, Handhabbarkeit und Wartbarkeit ankommt.

Um die Größe dieses Prozedurstacks bestimmen zu können, muß der Entwerfer von MIMOLA-Eingabebeschreibungen die maximale Schachtelungstiefe seiner Prozeduren kennen bzw. kann sie von dem Synthesesystem ermitteln lassen. Dementsprechend sind auch rekursive Beschreibungen mit Vorsicht zu handhaben.

Eine zweiter Nebeneffekt, der sich aus der Verwendung von Prozeduren ergibt, betrifft die Handhabung lokaler Variablen. So führt deren Verwendung im Syntheseergebnis evtl. zu zusätzlichen Registern, mit einem entsprechend erhöhten Hardwarebedarf. Entsprechend der Prozedurschachtelung und der Lebensdauer der Variablen ist es deshalb für die High-Level Synthese möglicherweise effizienter, wenn der Entwerfer in seiner Eingabedatei globale Variablen benutzt und so eine explizite Mehrfachbenutzung der Register erreicht.

Im Lauf des Projekts hat sich dabei sehr schön der Lerneffekt gezeigt: Die Fehler, die noch in der ersten Beschreibung zu `bincalc` enthalten waren, sind bei dem späteren Entwurf `bcdcalc` nicht mehr wiederholt worden.

Verwendete Arithmetik

In dem verwendeten Beispiel des Taschenrechners wurde bei dem ersten Entwurf `bincalc` eine BCD-ALU benötigt, die aus ihrer prozeduralen Beschreibung heraus nicht korrekt synthetisierbar war. Der Grund dafür ist, daß arithmetische Operatoren wie Addition und Subtraktion bei MIMOLA über vorzeichenlosen Integerzahlen (unsigned) oder Zahlen im 2-Komplement definiert sind. Bei davon abweichenden Zahlendarstellungen muß der betreffende Operator durch ein Makro ersetzt werden, das dann auf RT-Ebene (von Hand) in der Netzliste durch Gatterlogik ergänzt wird.

Um jetzt noch eine Simulation der Verhaltensbeschreibung durchführen zu können — was ja gerade einer der Vorteile der High-Level Synthese ist —, muß ein entsprechendes Verhaltensmodell für diesen Operator beschrieben werden. Der einfachste und auch von uns gewählte Weg dazu ist die Beschreibung als PLA.

Optimierungen durch MIMOLA

Einige Probleme mit dem Entwurf gab es durch Optimierungen, die MIMOLA während der Hardwaresynthese durchführt. So kam es bei ineinander geschachtelten `if ... then ... else ...` Konstrukten und teilweise bei Prozedurschachtelungen zu einem falschen Verhalten der synthetisierten Hardware. Der Grund dafür war, daß im Programmablauf — hier noch nicht als Mikroprogramm, sondern als Zwischendarstellung — mehrere Rücksprünge aufeinander folgten. Bei nachfolgenden Syntheseschritten, die dann zur eigentlichen

RT-Beschreibung führen, wurden, wenn mehrere Sprungbefehle aufeinander folgten, einige von ihnen als dead-code interpretiert und „wegminimiert".

Durch Änderungen der Optionen, die die Minimierung beeinflussen, ließ sich dieses Verhalten leider nicht abstellen. Das Einfügen von „Dummy-Operationen" in der MIMOLA Beschreibung führte schließlich dazu, das solche aufeinanderfolgenden Rücksprünge nicht mehr auftraten und die synthetisierte Struktur fehlerfrei arbeitete.

Simulation auf höheren Ebenen

Gerade das oben beschriebene Beispiel zeigt, wie wichtig es ist, die Simulation der Schaltung auf verschiedenen Ebenen während des Entwurfs durchzuführen. Wir haben dazu folgende Simulationsläufe gemacht

Pascal, MSSS: Im ersten Schritt wurde eine reine Verhaltensbeschreibung des Taschenrechners simuliert. Dazu haben die Studenten entweder einen Pascal-Compiler mit Debugger benutzen können, wenn sie die Algorithmen zuhause entwickelt haben, oder nach der Umsetzung in die MIMOLA-Eingabe konnte der Simulator MSSS benutzt werden.

MSSU: Anschließend wurde die von MIMOLA synthetisierte RT-Struktur simuliert. Diese Struktur besteht aber noch aus „generischen" Elementen, und ist noch nicht auf eine Zielbibliothek abgebildet worden.

Bei diesem Schritt konnte beispielsweise das fehlerhafte Verhalten nach der Optimierung festgestellt werden. Da der Simulator interaktiv arbeitet, ließ sich der Fehler vergleichsweise schnell lokalisieren.

MADS: Nach der Bearbeitung durch MIM2SOLO wurden noch abschließende Simulationen mit dem Simulator von SOLO 1400 vorgenommen. Dabei wurde dann die RT-Beschreibung aus den Elementen der Standarzellbibliothek simuliert. Erfreulicherweise sind bei den MADS Simulationen keinerlei Fehler mehr aufgetreten, so daß die Iterationsschritte, die zu einer funktional korrekten Schaltung führten, nur MIMOLA betrafen.

2.2.2 MIM2SOLO

Die Umsetzung der MIMOLA Ausgabe in die Eingabe von SOLO 1400 mit MIM2SOLO verlief ohne Probleme. Allerdings mußten die Ausgaben zum Teil noch nachbearbeitet werden; dies hatte folgende Gründe:

- Wegen Fehlern in SOLO 1400-Programmen (s.u.) wurden eigene Tools eingesetzt, die einen Workaround ermöglichten.

- Um die Testbarkeit der Schaltung zu gewährleisten, wurden der Scanpfad und die Testausgabe der ROMs „von Hand“ in die SOLO 1400-Eingabe eingearbeitet; ein Einbetten in die MIMOLA-Beschreibung schien nicht sinnvoll, da dies zu sehr ineffizienten Syntheseergebnissen geführt hätte — auch hier zeigt sich wieder, wie wichtig eine gewisse „Designerfahrung“ beim Umgang mit High-Level Synthesewerkzeugen ist.

2.2.3 SOLO 1400

Das letzte Glied in unserer Tool-Chain war das Standardzell-Entwurfssystem SOLO 1400, mit dem wir bislang, wegen seiner vergleichsweise einfachen Handhabbarkeit und der sehr effizienten Möglichkeit einer textuellen Schaltungseingabe, sehr gute Erfahrungen im Bereich der Lehre gesammelt haben. Mit den Werkzeugen in SOLO 1400 wurden die abschließenden Simulationen durchgeführt und das physikalische Layout (Placement & Routing) erstellt. Bei den in diesem Projekt durchgeführten Arbeiten zeigten sich aber doch noch einige Schwächen des Systems:

Fehler in der Software

Einige schon vorher bekannte Fehler in der Software führten dazu, daß zusätzliche eigene Programme eingesetzt werden mußten, um entsprechende Workarounds zu erzeugen. Wie es aussieht, sind in der Programmierung einige Datentypen streng auf 32 Bit Länge ausgerichtet.

Das führt dazu, daß, bei der Beschreibung der Hardwarestruktur auf RT-Ebene, Busse mit mehr als 32 Bit Wortbreite zu Problemen führen.

Für den Simulator von SOLO 1400, MADS, sind auch zwei solcher Phänomene bekannt. Zum einen zeigen ROMs mit mehr als 32 Bit Wortbreite in der Simulation ein falsches Verhalten, obwohl die Datei, die den Inhalt des ROMs und damit das Verhaltensmodell für die Simulation enthält, korrekt ist. Zum anderen kommt es bei langen Simulationsläufen zu einem Überlauf des Zählers für den Simulationszeitpunkt, so daß von da ab die Ereigniszeitpunkte nicht mehr stimmen.

Plazierung & Verdrahtung

Vom Konzept her ist SOLO 1400 als Entwurfssystem für Designs geringer oder mittlerer Komplexität (< 12 000 Gatter) gedacht, das dabei von Anwendern genutzt wird, die überwiegend aus anderen Aufgabengebieten kommen und vergleichsweise wenig Erfahrung mitbringen müssen. Dies erklärt die sehr einfache Bedienbarkeit, die SOLO 1400 für den Einsatz in der Lehre auszeichnet, führt aber auch dazu, daß gerade im Bereich der Plazierung & Verdrahtung relativ simple Algorithmen eingesetzt werden. So ist es zwar vollautomatisch möglich, ein Chiplayout zu generieren, die erzielten Ergebnisse sind jedoch qualitativ (benötigte Fläche) denen anderer Entwurfssysteme unterlegen — als Beispiel für ein „schlechtes“ automatisch generiertes Layout sei auf Abbildung 2 verwiesen. Außerdem sind die Eingriffsmöglichkeiten in die Plazierung & Verdrahtung stark beschränkt — was im Sinne einfacher Handhabung erwünscht ist, jedoch in Problemfällen genau den gegenteiligen Effekt hat: kommt es wegen der Größe des Entwurfs zu Problemen mit der Plazierung und Verdrahtung der Standardzellen, so läßt sich ein „korrektes“ Layout nur noch mit sehr großem Arbeitsaufwand erstellen.

Um die Korrektheit des Layouts im Sinne der Schaltungstechnik zu gewährleisten, gibt es in SOLO 1400 mehrere Testprogramme, die überprüfen ob:

- die Verteilung der Versorgungsspannung gewährleistet ist (Anzahl und Anschluß der Power- und Ground-Pads).
- die Treiberleistung aller Zellen ausreichend ist, um nachfolgende Eingänge zu speisen.
- die Länge von Leitungsnetzen nicht zu groß ist, da es sonst zu Problemen mit der Treiberleistung und der zeitlichen Verzögerung der Signale auf den Leitungen kommen kann.

Gerade die letzten beiden Punkte führten bei unserem Entwurf zu unerwarteten Problemen. So mußten bei einigen stark belasteten Netzen (z.B. der zentralen `clock`) zusätzliche Treiber eingefügt werden — was eigentlich noch sehr einfach ging. Hinzu kam aber, daß sehr viele Netze zu lang waren, was uns extreme Probleme bereitete. Da SOLO 1400 nicht über eine vernüftige graphische Ausgabe der Plazierungs- und Vedrahtungsergebnisse verfügt, war es sehr schwierig, die betreffenden zu langen Netze an definierten Stellen durch Treiber in kürzere zu unterteilen. Außerdem trat bei dieser „Nachbearbeitung“ folgender Zyklus auf: Bei der Plazierung und Verdrahtung einer gegebenen Netzliste sind zu große Netze aufgetreten – dementsprechend werden Treiber eingebaut und die Netze unterteilt – die so entstandenen Netzliste wird erneut plaziert und verdrahtet – durch das geänderte Layout sind andere Netze zu groß geworden...Insgesamt kam es dabei zu einem Iterationsprozeß, der für das Generieren eines „abgabefertigen“ Layouts mehrere Tage dauerte.

Dazu muß noch gesagt werden, daß der Entwurf `bcdcalc` von seiner Komplexität noch nicht an die Grenzen von SOLO 1400 stößt, vielmehr scheinen die hier aufgetretenen Probleme mit architekturellen Eigenschaften von mit MIMOLA synthetisierten RT-Beschreibungen zusammenzuhängen:

- Aufgrund der Bus- und Multiplexerstrukturen kommt es zu stark verzweigten Datenpfaden und dementsprechend zu großen Netzen im Layout.
- Um Steuerleitungen einzusparen, versuchen die in MIMOLA verwendeten Algorithmen die Steuerleitungen mehrfach zu benutzen — wie auch bei allen anderen Synthesesystemen, die mikroprogrammierte Architekturen erzeugen. Da die Steuerwortbelegungen im Regelfall sehr viele don't-care Stellen besitzen, gelingt das entsprechend gut, führt aber logischerweise dazu, daß Ausgänge des Mikroprogrammspeichers an sehr vielen Stellen im Operationswerk benutzt werden und deshalb nach der Verdrahtung des Layouts sehr lange Netze entstehen.

2.2.4 Zusätzlich notwendige Werkzeuge

Wie schon oben angesprochen, waren wir durch Fehler in den SOLO 1400 Programmen gezwungen, zusätzliche eigene Werkzeuge einzusetzen, die Workarounds in die RT-Beschreibung einarbeiteten.

Aufsplitten von Bussen

So dient ein eigenes Programm `splitbus` dazu, über einen Textersetzungsmechanismus Busse mit mehr als 32 Bit Breite in kleinere Busse aufzuteilen.

Verringerung der Mikroprogrammwortlänge

Obwohl der Fehler bei zu großen Wortlängen von ROMs nur das Verhalten in der Simulation zu betreffen schien — die Ausgaben des ROM Generators waren korrekt —, war eine Fertigung der Schaltung ohne eine entsprechende Simulation mit „realen" Elementen der Zellbibliothek von vornherein ausgeschlossen. Prinzipiell gab es zwei alternative Lösungsansätze, um diesen Fehler zu umgehen.

Zum einen konnte das ROM, analog der Aufteilung von Bussen, in mehrere kleinere Speicher unterteilt werden, deren Inhalte konkateniert das ursprüngliche Steuerwort ergeben. Dazu gab es schon Programme, die wir auch schon bei früheren Entwürfen erfolgreich eingesetzt hatten. In einer ersten Version von `bcdcalc` wurde der Mikroprogrammspeicher in drei ROMs realisiert.

Über diesen Ansatz hinausgehend schien, bei mit MIMOLA synthetisierten Mikroprogrammspeichern, der Ansatz der Aufteilung mit geichzeitiger Minimierung der Wortlänge erfolgversprechend. Wie schon oben beschrieben, versucht MIMOLA die Steuerwortlänge durch Ausnutzung von don't-care Stellen zu reduzieren. Als Randbedingung gilt dabei allerdings, daß keine zusätzliche Hardware investiert wird, also nur die Überdeckung von Steuerbits ausgenutzt werden kann. Auch nach der Synthese durch MIMOLA enthält der Mikroprogrammspeicher überwiegend don't-cares. Deshalb wurden während dieses Projekts zwei zusätzliche Programme `minrom` und `diffrom` entwickelt [Mäder 93], die eine Minimierung nach folgenden Kriterien vornehmen — der Mikroprogrammspeicher sei hier als eine Matrix beschrieben:

- Zuerst werden konstante Belegungen von Steuerleitungen, entsprechend konstanten Spalten, eliminiert.
- Dann werden identische Spalten und Spalten, die sich durch Ausnutzen von don't-care Stellen überdecken lassen, zusammengefaßt. Da MIMOLA Steuerwortteile entsprechend ihrer Verwendung im Operationswerk in logisch zusammengehörige Gruppen unterteilt, treten auch diese Fälle noch auf.
  Diese ersten beiden Maßnahmen verringern die Länge eines Steuerwortes, ohne zusätzlichen Aufwand in der Hardware zu erzeugen.

– In dem nun folgenden Schritt wird versucht, Spalten durch logische Kombinationen (UND, ODER, NICHT) anderer Spalten (unter Ausnutzung von don't-cares, auch mehrstufig) zu ersetzen.
– Abschließend wird nach Möglichkeiten für Gruppenkodierungen gesucht und die gefundenen Steuerleitungen durch das Codewort und eine nachgeschaltete Dekodierlogik ersetzt.

Das zweite Programm `diffrom` verifiziert die so gefundenen Ergebnisse und generiert die Eingabe zu SOLO 1400, bestehend aus einem oder mehreren „kleinen" ROMs, mit jeweils weniger als 32 Bit Wortbreite, und dem Dekodierungsschaltnetz.

Da das Auffinden einer „minimalen" Lösung NP-vollständig ist, kommen Heuristiken zum Einsatz, die sich aber bei den bislang von uns verwendeten Beispielen als sehr effizient erwiesen. So konnte beispielsweise das in `bcdcalc` von MIMOLA generierte ROM von 652 Worten á 76 Bit auf 38 Bit — realisiert in zwei ROMs 652×19 Bit — reduziert werden. Die Flächenersparnis durch das/die kleineren ROMs überwiegt dabei deutlich gegenüber dem Aufwand für die Zusatzlogik. Auch vom Zeitverhalten ist diese Lösung etwas schneller, da kleinere ROMs kürzere Zugriffszeiten haben und so die zusätzliche zeitliche Verzögerung in der Nachfolgelogik kompensiert wird.

2.3 Alternativen

Der letzte Abschnitt unseres Erfahrungsbericht befaßt sich mit Alternativen zu den hier im Projekt verwendeten Werkzeugen MIMOLA und SOLO 1400.

Synthese

Zu Beginn des Projektes hatten wir für die High-Level Synthese neben MIMOLA auch noch das Olympus System [Ku & Micheli 90] zur Verfügung, haben es aber aus folgenden Gründen nicht eingesetzt:

– Die Aufgabenstellung war so gewählt, daß sie auch mit MIMOLA synthetisierbar war; die Beschränkung auf ein zentrales Steuerwerk stellte kein Problem dar.
– Die Pascal ähnliche Eingabesprache MIMOLA konnte sehr schnell erlernt werden.
– Mit MIM2SOLO verfügten wir über eine in mehreren Entwürfen erprobte Anbindung der High-Level Synthese an ein Standardzell-Entwurfssystem.

Inzwischen haben wir über das ESPRIT Projekt EUROCHIP auch den „de facto" Standard bei den industriellen Synthesesystemen SYNOPSYS erhalten. Zu der Eignung von SYNOPSYS für solch ein Projekt ließe sich sagen:

Eingabesprache: Mit VHDL verfügt SYNOPSYS über die Standard-Hardwarebeschreibungssprache. Von daher wäre ein Einsatz in der Lehre wünschenswert und wird längerfristig auch erfolgen. Wegen des Sprachumfangs — Beschreibungsmöglichkeiten von „reinem" Verhalten, über Mischformen bis hin zu Strukturbeschreibungen auf unteren Entwurfsebenen — und einer vergleichsweise komplizierten, zum Teil nicht formal spezifizierten und stark simulationsbezogener Semantik, muß der Einsatz in der Lehre über mehrere Veranstaltungen vorbereitet werden. Die Einarbeitung in VHDL während eines Projekts ist vom Arbeitsaufwand her nicht möglich.

Außerdem tritt gerade bei VHDL-Verhaltensbeschreibungen der schon vorher angesprochene Effekt auf, daß ein Entwerfer über viel Erfahrung beim Umgang mit dem Werkzeug vefügen muß, um eine „gut synthetisierbare" Eingabe zu erstellen. Der Grund dafür liegt in den vielen Möglichkeiten, äquivalentes Verhalten in VHDL auszudrücken.

Mächtigkeit des Werkzeugs: Als industrielles Werkzeug verfügt SYNOPSYS über entsprechende Schnittstellen zu Standardzellentwurfssystemen und kann auch die Abbildung der Hardwarestruktur auf Zellbibliotheken vornehmen.

Andererseits sind Werkzeuge, die aus dem universitären Umfeld kommen — dem Stand der Forschung entsprechend —, oft mächtiger in Bezug auf das, was als „High-Level Beschreibung" synthetisiert werden kann und implementieren bessere Algorithmen als kommerzielle Tools. Entsprechend einer ersten qualitativen Bewertung ist SYNOPSYS ein sehr gutes Werkzeug zur Logiksynthese und bietet eingeschränkte Möglichkeiten zur High-Level Synthese.

Standardzellentwurf

Für den Bereich des Standardzellentwurfs verfügen wir am Arbeitsbereich noch über CADENCE/SOLO 2030, das sich gegenüber SOLO 1400 durch folgende Eigenschaften auszeichnet:

+ CADENCE/SOLO 2030 ist ein offenes und universelles Entwurfssystem: Es können beliebige Zellbibliotheken und Prozeßtechnologien benutzt werden, weiterhin ist auch der Full-Custom Entwurf mit dem System möglich und zusätzliche Werkzeuge (z.B. Simulatoren) können integriert werden.

+ Die Größe der zu entwerfenden Schaltung ist nicht durch Eigenschaften der Software, sondern höchstens durch die Rechenleistung und den Speicherplatz der verarbeitenden Maschine limitiert.

+ Die bei der Plazierung & Verdrahtung von Standarzellen verwendeten Algorithmen liefern sehr gute Ergebnisse und erlauben vielfältige Eingriffsmöglichkeiten.

+ Das System ist sowohl in seiner Bedienung, als auch in dem Erscheinunsbild seiner Oberfläche beliebig konfigurierbar.

+ Die integrierte Programmiersprache SKILL erlaubt die Verwendung von Benutzereigenen Algorithmen.

– In seiner zum Projektzeitpunkt vorliegenden Version verfügte CADENCE/SOLO 2030 allerdings noch nicht über eine zuverlässig arbeitende Form der textuellen Schaltungseingabe; die Schnittstelle nach außen war dementsprechend das Schematic-Entry.

– Außerdem bewirken die oben genannten Punkte — Mächtigkeit und Universalität von CADENCE/SOLO 2030 —, daß die Benutzung des Systems viel Erfahrung erfordert, da man sehr schnell in der Konfiguration etwas falsch eigestellt hat und daß die Einarbeitung sehr lange dauert.

  Auch hier, wie bei VHDL, bedarf es einer längerfristigen Vorbereitung über mehrere Veranstaltungen, um einen Einsatz in der Lehre zu ermöglichen. Im Rahmen des Projekts `18.332` war unsere Intention, daß die Studenten den Entwurfsablauf bei der High-Level Synthese begreifen und an einem Beispiel durcharbeiten können. Da der Schwerpunkt eindeutig beim „Design-Flow“ und Inhalten der Aufgabenstellung liegt, sollte bei dem ohnehin knappen Zeitrahmen eines Semesters der Arbeitsaufwand nicht zum Erlernen von Werkzeugen eingesetzt werden.

Da der Bereich der Plazierung & Verdrahtung mit SOLO 1400 unerwartet Probleme bereitete, haben wir über zusätzliche eigene Programme, die als Bibliotheksumsetzer und Schematic-Generatoren arbeiteten, die Eingabe für CADENCE/SOLO 2030 erzeugt. Da beim Entwurf `bcdcalc` ein Großteil der Chipfläche für die generierten Blöcke (RAMs und ROMs) verwendet wird, ist die Einsparung an Chipfläche durch die besseren Plazierungs- und Verdrahtungsalgorithmen, trotz Reduktion auf ca. 70 % (Core Bereich: 28,7 mm^2 bei SOLO 1400 zu 20 mm^2 mit CADENCE/SOLO 2030), noch nicht so groß wie bei Entwürfen, die überwiegend aus Standardzellen aufgebaut sind.

Da aber auch das CADENCE/SOLO 2030 Layout noch zu lange Netze enthielt, die einer Nachbearbeitung bedurft hätten, und da zu diesem Zeitpunkt das Layout mit SOLO 1400 quasi schon fertig war, ist der zur Fertigung eingereichte Entwurf letztendlich doch mit SOLO 1400 erstellt worden.

Resümee

Wie schon bei den Werkzeugen angesprochen, wird es in Zukunft einen durchgängigen Pfad geben: ausgehend von VHDL-Spezifikationen — über SYNOPSYS für die Logiksynthese und eingeschränkt für die High-Level Synthese — zu CADENCE/SOLO 2030 für den Standardzellentwurf. Wir planen in späteren Projekten diese Werkzeuge einzusetzen, allerdings immer mit der Einschränkung, daß die Studenten durch vorherige Praktika und Projekte schon Erfahrungen im Umgang mit diesen Tools gesammelt haben.

3 bincalc: Ein Taschenrechner mit binärer Arithmetik

Der erste im Laufe des Projekts von Studenten erstellte und bis zu einem Chiplayout entworfene Taschenrechner, `bincalc`, basiert auf einer ungewöhnlichen Kombination von binärer und BCD Arithmetik.

3.1 Beschreibung

Die Studenten waren mit dem zu Beginn des Projekts vorgeschlagenen Konzept eines einfachen Rechners mit Festkommazahlen und BCD Arithmetik nicht zufrieden. Statt dessen entwarfen Sie einen Rechner, der intern eine dem IEEE 754 Format ähnliche Zahldarstellung mit 32 Bit Mantisse und 10 Bit Exponent mit Offset 512 benutzt.

Dadurch wird dann natürlich eine Konvertierung binär-dezimal und dezimal-binär notwendig. Die Umrechnung der Tasteneingaben (dezimal) in das binäre Format ist sehr leicht realisierbar und erfolgt durch die Multiplikation des aktuellen Eingaberegisters mit 10 und nachfolgende Addition der eingegebenen Ziffern.

Dagegen ist die Umrechnung der binär dargestellten Rechenergebnisse in eine für die Anzeige geeignete Form weit aufwendiger. Die Spezifikation von `bincalc` sieht dazu zwei Register `display` und `double` mit BCD kodierten Ziffern und einer eigenen BCD-ALU vor:

Zu Anfang der Umrechnung wird das BCD Register `double` mit der Konstante 1 geladen und durch wiederholtes Verdoppeln/Halbieren mit der BCD-ALU auf den Wert 2^{i-512} gebracht, wobei i den Wert des Exponenten in der Binärdarstellung bezeichnet. Dann wird dieser Wert von `double` nach `display` geladen. Anschließend wird für jedes Bit der Mantisse `double` halbiert, und wenn das entsprechende Bit der Mantisse Eins ist, der Wert von `double` zu `display` addiert. Obwohl sich bei dieser Umrechnung Rundungsfehler akkumulieren können, wird die angestrebte Genauigkeit von 8 Dezimalstellen erreicht.

Im einzelnen umfaßt `bincalc` folgende Funktionen:

- Taschenrechner-CPU für Fließkommazahlen mit binärer Zahlendarstellung mit 32 Bit Mantisse (normalisiert mit führender, impliziter Eins) und Vorzeichenbit, Exponent mit 10 Stellen und Offset 512
- gemultiplexte Sieben-Segment Anzeige, gemultiplexte entprellte Tasteneingabe
- serielle binäre Arithmetik
- vier Grundrechenarten (Addition, Subtraktion, Multiplikation, Division)
- Algebraische Eingabelogik (Infix-Notation)
- 1 Speicherregister mit `M+` und `M*` Operationen

3.2 Erfahrungen beim Entwurf

Das erste funktionsfähige Pascal-Programm für bincalc umfaßte etwa 1600 Zeilen Code. Durch die häufige Verwendung von vielen Pascal-typischen, aber für die Hardwaresynthese eher problematischen Konstrukten, erwies sich die Umsetzung in eine brauchbare MIMOLA Spezifikation als unerwartet schwierig.

Die meisten dieser Probleme können bei der Erstellung einer für eine Synthese geeigneten Spezifikation wohl als typisch gelten:

- Die anfängliche bincalc Spezifikation verwendete in bestem Pascal Programmierstil eigene Datentypen für alle vorkommenden Variablen.

  Dies ist beim Einsatz von MIMOLA zunächst kein Problem, da MIMOLA auch Arrays und Records als Datentypen kennt und synthetisieren kann. Allerdings muß ein Overhead, etwa durch die komplexere Adressberechnung zum Zugriff auf einzelne Komponenten von Records in Speichern, in Kauf genommen werden. Dies ist auf normalen von-Neumann Rechnern mit ihrem großen Speicher und geringen Performanz-Anforderungen oft kein wichtiger Punkt:

  ```
  TYPE
     Dual = RECORD
               Mantisse: ARRAY[0..MaxMant] OF BIT;
               Exponent: ARRAY[0..MaxExpo] OF BIT;
            END;
  VAR
     X,Y,Z: Dual;
  ```

  Ein Zugriff etwa auf X.Exponent[j] erfordert dann die Berechnung einer Speicheradresse gemäß

  ```
  Addr& = baseaddr(X) + size(Mantisse) + j*size(BIT)
  ```

  Für bincalc ergab sich jedoch durch den aufwendigen Zugriff auf einzelne Komponenten der Registervariablen ein erheblicher Zusatzaufwand. Die MIMOLA Synthese erzeugte zwei zusätzliche Addierer, um in einem Mikroinstruktionszyklus auf ein Bit i etwa im X-Register zugreifen zu können.

- Auf einige der verwendeten Datentypen von bincalc wird sowohl in Teilen als auch gesamt zugegriffen. Die ursprünglich vorgesehene Realisierung der Rechenregister als Register mit 32 Bit Breite (nur Mantisse) konnte nicht realisiert werden, da viele der Algorithmen einzelne Bits der Mantisse schreiben. Dies hätte sehr aufwendige Multiplexer vor den Registern erfordert und schloß außerdem die Unterbringung dieser Rechenregister in entsprechend 32 Bit breiten Speichern aus.

Da auch auf einzelne Bits der Exponenten zugegriffen wird, erschien die Abbildung der Rechenregister auf 1 Bit breite Speicher noch am besten. Da für die vorgesehene Anzahl von Rechenregistern insgesamt gerade 240 Bit benötigt wurden, konnte ein Speicher der Größe 256×1 Bit eingesetzt werden. Die BCD-Register `double` und `display` wurden entsprechend in einem weiteren RAM der Größe 32×4 untergebracht.

- Im Rahmen eines Top-Down Entwurfs mit schrittweiser Verfeinerung ist die Aufteilung des gesamten Programms in einzelne Prozeduren natürlich unbedingt notwendig.

  Wenn die Synthesewerkzeuge diese Prozeduren aber nicht alle „inline" realisieren sollen oder können, bedingt dies einen Stack zur Speicherung der Rücksprungadressen. Für `bincalc` wurde schließlich ein Stack mit acht Einträgen gewählt und in einem weiterem RAM mit 8×10 Bit untergebracht — dies reicht auch bei der tiefsten vorkommenden Schachtelung in der Division gerade aus.

- Alle Prozeduren in der ersten `bincalc` Spezifikation verwendeten lokale Variablen der verschiedenen Datentypen. Zwar kann MIMOLA über die Verwendung von Replacement-Rules einen Stack verwalten und damit lokale Variable bereitstellen. Dies hätte aber über den Speicher zur Verwaltung der Rücksprungadressen hinaus einen weiteren Speicher bedeutet, da die im Registerspeicher verbliebenen freien 16 Bit (256 - 240) nicht für die lokalen Variablen ausgereicht hätten. Eine Verdoppelung der Größe des Registerspeichers erschien als zu teuer.

  Deshalb wurde versucht, die meisten dieser (Hilfs-) Variablen durch Änderung der Spezifikation zu beseitigen oder über globale Variable zu realisieren. Die Anzahl der benötigten zusätzlichen Variablen konnte mit diesem — allerdings mühsamen — Verfahren drastisch reduziert werden, so daß schließlich nur noch vier 1 Bit Variable und zwei 4 Bit Variable benötigt wurden.

- Bei der Beschreibung der für die Umrechnung binär-dezimal nötigen BCD-ALU ergaben sich ebenfalls einige Probleme.

  Die in Pascal bequemste Beschreibung über eine kleine Prozedur ergab natürlich keine effektive Hardwarerealisierung. Außerdem war die Beschreibung des Verhaltens der ALU in MIMOLA unerwartet schwierig. Die Deklaration der ALU als Dummy kam nicht in Frage, da dies die RT-Simulationen verhindert hätte.

  Schließlich wurde die BCD-ALU sowohl in Pascal als auch in MIMOLA über eine PLA-Tabelle beschrieben.

3.3 Ergebnis und Beschreibung des Chips

Nach Abschluß der High-Level Modifikationen der `bincalc` Spezifikation wurde dann probehalber mit MIM2SOLO eine MODEL Beschreibung und daraus ein SOLO 1400 Layout erzeugt. Beim ersten Versuch kamen dabei die BUS-style Multiplexer zur Anwendung, die in SOLO 1400 leider sehr ineffizient sind. Ein zweiter Entwurf mit einfachen Multiplexern führte zu einem automatisch generierten Layout mit allerdings noch sehr ungünstig plazierten Makrozellen von 54mm^2, das in Abbildung 2 gezeigt ist.

Dieser Entwurf verwendet vier Makrozellen, die zusammen mehr als 50% der Chipfläche einnehmen: Das Mikroprogramm benötigt ein ROM der Größe 1024×80 Bit — wobei wegen Fehlern in den SOLO 1400 internen Programmen noch eine Aufspaltung in kleinere ROMs notwendig wäre —, und den drei RAMs mit 256×4, 32×4 und 8×10 Bit für die Rechenregister, die BCD-Register und den Prozedurstack. Die Verwendung eines RAMs mit 256×4 Bit (anstelle von 256×1 Bit) wurde nötig, da die ES2 RAM-Generatoren nur RAMs mit mindestens 4 Bit Breite erzeugen können.

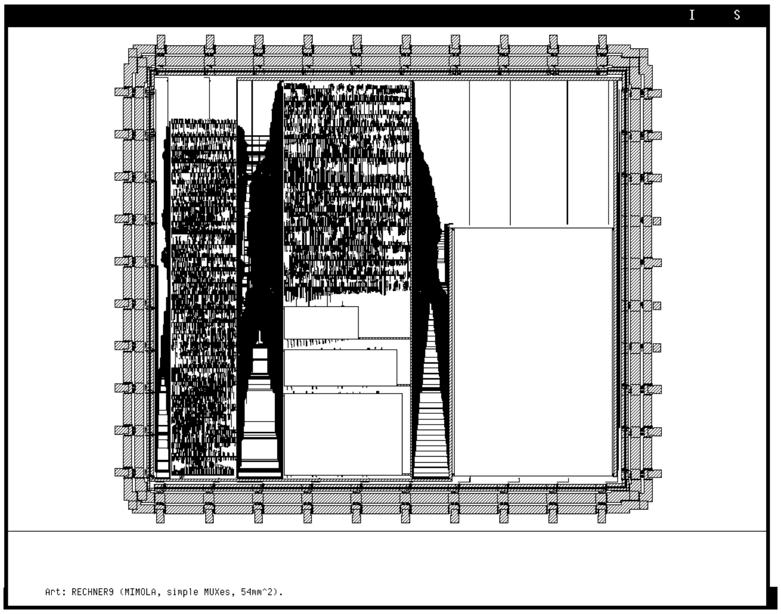


Abbildung 2: Automatisch generiertes Layout von `bincalc`, keine Fertigung

4 bcdcalc: Ein Taschenrechner mit BCD Arithmetik

Nach unserem ersten Beispielentwurf und dem Design mit binärer Arithmetik `bincalc`, umfaßt der dritte im Laufe des Projekts entstandene Entwurf, `bcdcalc`, den vollen Funktionsumfang eines Fließkomma-Taschenrechners mit algebraischer Eingabelogik („infix"), Operatorpriorität („punkt vor strich") und 10 Klammerebenen. Dieser Entwurf wird über EUROCHIP gefertigt werden.

Das Pascal-Programm und die MIMOLA Beschreibung für `bcdcalc` wurden weitgehend vom Studenten Stefan Eggers erstellt. Durch die Verwendung von BCD Arithmetik für alle Funktionen und eine geschickte Unterbringung und Adressierung aller Rechenregister in einem Speicher ist `bcdcalc` weit effektiver als die vorherigen Entwürfe. Die geplante Erweiterung um höhere Funktionen (Quadratwurzel, Exponentialfunktionen, Winkelfunktionen etc.) konnte aus Zeitgründen bisher nicht realisiert werden.

4.1 Beschreibung

Die Pascal/MIMOLA Spezifikation von `bcdcalc` erweitert die Möglichkeiten der früheren Entwürfe stark. Im einzelnen umfaßt `bcdcalc` folgende Funktionen:

- Taschenrechner-CPU für Fließkommazahlen mit 10 Stellen Mantisse und zwei Stellen Exponent,

  Zahlenbereich $\pm 9.999\,999\,999^{\pm 99}$
- gemultiplexte Sieben-Segment Anzeige, gemultiplexte entprellte Tasteneingabe
- serielle BCD Arithmetik
- vier Grundrechenarten (Addition, Subtraktion, Multiplikation, Division)
- Algebraische Eingabelogik (Infix-Notation),

  rekursives Löschen der letzten Operation
- Operatorpriorität (Punkt-vor-Strich, Exponentiation vorgesehen)
- 10 Klammerebenen
- 1 Speicherregister mit `M+` und `M*` Operationen
- Fehlerbehandlung für Überlauf und Division durch Null, „gradual underflow"

Da der Entwurf zur Fertigung eingereicht werden sollte, haben wir, neben den oben aufgezählten Funktionen, zusätzliche Logik implementiert, um die Testbarkeit des ICs zu erhöhen, und um einen fehlerfreien Betrieb zu gewährleisten. Diese Testlogik ist dabei zum Teil schon im MIMOLA Programm beschrieben (Speicherselbsttest) und synthetisiert worden, während „klassische“ Methoden, wie Scanpath und Ausgabe des ROM-Inhalts erst später in der MODEL Datei ergänzt worden sind:

- Speicherselbsttest bei Inbetriebnahme
- Testmodus zur Umgehung der Zeitschleifen bei Ein- und Ausgabe (direkte Eingabe über eigenen Bus, einmalige Ausgabe des Ergebnisses)
- Scanpath
- Auslesemöglichkeit des Mikroprogrammspeichers

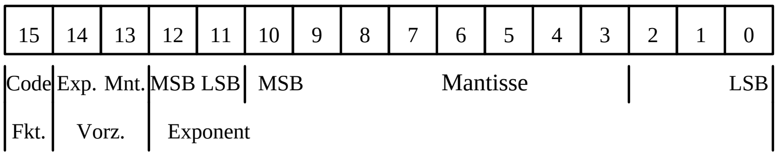
Eine der wichtisten Verbesserungen von `bcdcalc` gegenüber `bincalc` betrifft die Verwendung eines zentralen RAM zur Speicherung aller Variablenwerte und die Zuordnung der einzelnen Register/Variablen zu Adressen in diesem RAM. `bcdcalc` verwendet dazu ein RAM der Größe 256×4 Bit, das in 16 Register mit je 16 BCD-Digits aufgeteilt wird. Jedes Register wiederum ist in 11 Digits der Mantisse, 2 Digit Exponent, 2 Digits Vorzeichen und ein „Funktions“digit aufgeteilt, siehe Abbildung 3. In den Funktionsdigits wird die zwischen Register i und $i-1$ auszuführende Operation gespeichert, wenn diese wegen Klammerung oder niedriger Operatorpriorität nicht sofort ausgeführt werden kann.

Als Beispiel für die Art der Zahldarstellung ist in Abbildung 4 die Ausgabe von `bcdcalc` für die Aufgabe $(6-8.7)/7$ dargestellt (im Testmode, so daß die Ausgabe der Werte nur einmal erfolgt). Die Interpretation der vom Sieben-Segment Dekoder gelieferten Werte `A4 D6 FE BA 74 24 A4 D6 FE B6 24 EE 10 10 (10)` ergibt die Ziffernfolge `7 5 8 2 4 1 7 5 8 3 1 0 - -`, die in die richtige Reihenfolge umsortiert das Ergebnis $-3.857\,142\,857^{-01}$ liefert.

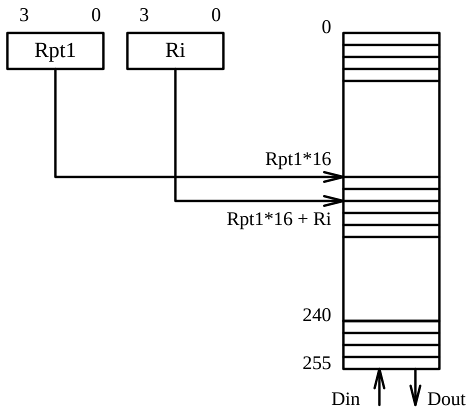
Zwei Register, nämlich R14 und R15, haben spezielle Funktionen, die anderen Register werden mit R0 und R1 beginnend für den Stack verwendet. R14 dient als Speicherregister und R15 nimmt Zwischenergebnisse bei der Multiplikation und Division auf. Für zukünftige Erweiterungen um höhere Funktionen von `bcdcalc` ist vorgesehen, weitere Register für Zwischenergebnisse zu reservieren.

Adressen für den Speicher lassen sich damit sehr einfach durch Konkatenation zweier 4-bit Werte bilden, wobei die oberen 4 bit ein Register adressieren und die unteren 4 bit ein Digit in diesem Register.

`bcdcalc` verwendet drei globale Pointer: `Rpt0` und `Rpt1` zum Zugriff auf die aktuellen X- und Y-Register, und `Rpt2` zum Zugriff auf das letzte Ergebnis. Push- und Pop-Operationen für den Operandenstack, die zum Beispiel über die Operatorpriorität oder Klammeroperationen erforderlich werden, lassen sich daher elegant und einfach über Incrementieren/Decrementieren dieser Pointer realisieren.



Bedeutung der Digits in einem Register



Speicheradressierung

Abbildung 3: Registeradressierung in `bcdcalc`

4.2 Erfahrungen beim Entwurf

Durch den Verzicht auf lokale Variablen und die durchgehende Verwendung von MIMOLA-typischen Konstrukten — insbesondere für die Adressrechnung des Registerspeichers — war die Umsetzung des `bcdcalc` Pascal-Programms in eine MIMOLA Spezifikation sehr einfach. In die Modulbibliothek für die Hardwaresynthese wurden neben die Register, ALUs und Multiplexer wiederum ein-Port RAMs, der Sieben-Segment Dekoder und der 4-nach-16 Dekoder aufgenommen. Die MIMOLA-Datei umfaßt etwa 1900 Zeilen — 650 Zeilen für die Modulbibliothek und die Replacement-Regeln und gut 1200 Zeilen für den `bcdcalc` Algorithmus.

Die High-Level Synthese mit MIMOLA gelang sofort, allerdings zeigten sich bei den RT-Simulationen zwei Fehler, die im Pascal-Programm nicht auftraten:

- Der erste Fehler — falsche Rechenergebnisse bei einigen Aufgaben — trat zuerst bei

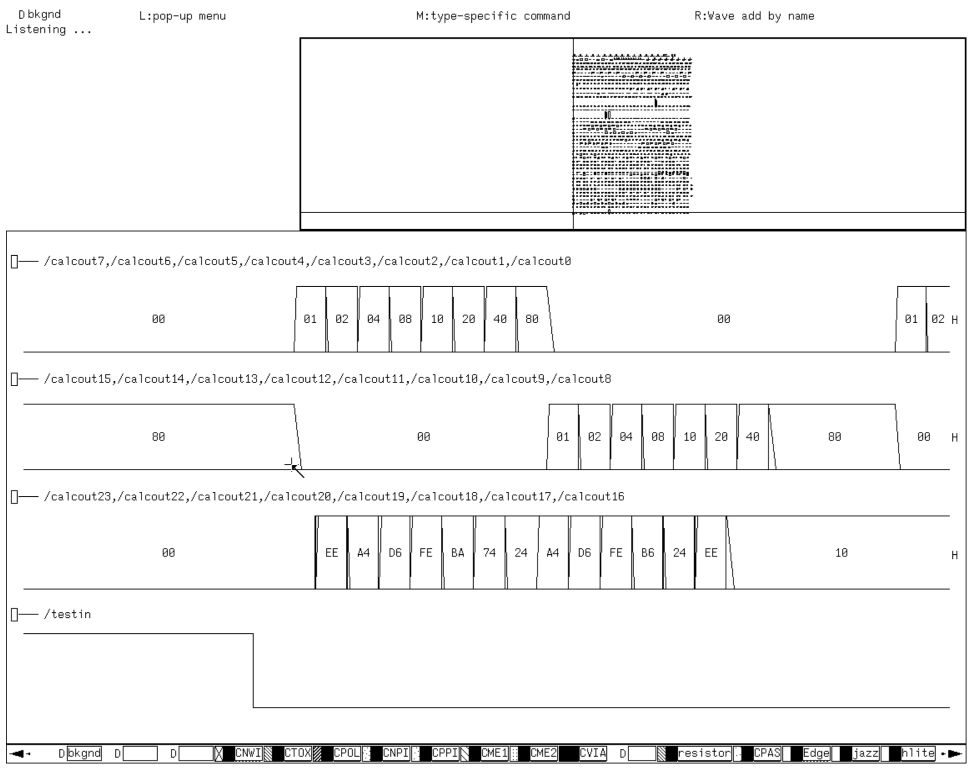


Abbildung 4: Rechenergebnis von `bcdcalc` (Gate-level Simulation)

einem unserer Testbeispiele auf, bei dem negative Zahlen vorkamen. Das Problem konnte schnell auf eine subtile Optimierung in der Komponente MSSI von MIMOLA zurückverfolgt werden — Prozeduraufrufe aus tief geschachtelten IF-THEN-ELSE Statements heraus führten manchmal zu einer fehlerhaften dead-code Elimination durch MSSI und damit zu falschen Rücksprungadressen. Der Fehler konnte durch die Einführung von dummy-Zuweisungen an den entsprechenden Stellen des MIMOLA-Programms umgangen werden. Alle folgenden RT-Simulationen lieferten dann auf Anhieb die erwarteten und korrekten Ergebnisse.

- In der Prozedur zur Fehlerbehandlung (Überlauf, Division durch Null) wurden noch zwei kleine Bugs entdeckt: Die Fehlerbehandlung setzte das Error-Flag nicht zurück, so daß der Rechner nach jedem Fehler nur durch einen Reset zu reaktivieren war.

  Dieses Beispiel zeigt besonders deutlich, daß Simulationen auf jeder Ebene nötig sind: Aufgrund des in der Fehlerbehandlung verwendeten nichtlokalen GOTO (mit direkter Modifikation des Prozedur-Stackpointers) konnte diese Prozedur in Pascal so nicht beschrieben werden und war durch eine — eben doch nicht völlig identische — Version ersetzt.

Als letzte Modifikation der High-Level Spezifikation wurde ein Speicherselbsttest in die Initialisierungsprozedur aufgenommen, der nach jedem Einschalten/Reset den Registerspeicher auf Fehler überprüft. Anstelle eines üblichen „Built-In Self-Test“ mit teuren LFSR-Registern und zusätzlich notwendiger Kontrollogik konnte der Selbsttest so mit nur 16 zusätzlichen Mikroinstruktionen realisiert werden.

Die anschließende High-Level Synthese mit MIMOLA und die Synthese in eine RT-Struktur mit MIM2SOLO verliefen dann problemlos. Die von MIM2SOLO erzeugten Module für den Sieben-Segment Dekoder und den 4-nach-16 Dekoder wurden allerdings in einem weiteren Schritt, ebenso wie der verwendete Komparator, durch effizientere Lösungen ersetzt.

Außerdem konnte das Mikroprogramm durch den Einsatz eines neuen Kompaktierungsprogramms `minrom` [Mäder 93] entscheidend verkleinert werden. `minrom` liest das Mikroprogramm unter Berücksichtigung von don't care Positionen ein und versucht, Spalten durch UND/ODER/NOT Verknüpfung anderer Spalten zu überdecken oder Gruppenkodierungen zu finden. Für das benötigte Dekodierungsschaltnetz wird anschließend eine MODEL Netzliste generiert.

Das von MIMOLA erzeugte horizontale Mikroprogramm mit 652 Instruktionen á 76 Bit konnte von `minrom` auf eine Breite von lediglich 38 Bit reduziert werden, bei minimalem Aufwand für das Dekodierungsschaltnetz. Die zusätzliche Verzögerung durch das Schaltnetz wird durch die schnellere Zugriffszeit des kleineren ROM aufgewogen.

Bei den anschließenden Gate-Level Simulationen mit dem SOLO 1400 Logiksimulator MADS wurden keine weiteren Fehler mehr entdeckt.

4.3 Ergebnis und technische Daten des **SOLO 1400**-Chips

Das aus der Synthese und dem Layout mit SOLO 1400 entstandene Chip ist in Abbildung 5 dargestellt. Die Gesamtfläche beträgt 43,96 mm^2 für den ecpd15 (1.5μm CMOS) Prozeß, wovon der eigentliche Core 29,95 mm^2 belegt und der Rest auf die Anschlußpads entfällt.

Der Entwurf besteht aus 2 144 Standardzellen mit zusammen circa 25 000 Transistoren und 4 Makrozellen. In dem Layout sieht man in der linken unteren Ecke die beiden RAMs: 8×10 Bit für den Prozedurstack und 256×4 Bit als Speicher für die BCD-Ziffern — das zugehörige Adressierungsschema wurde am Anfang dieses Kapitels vorgestellt. Die rechte obere Ecke des Core-Bereichs enthält die beiden ROMs mit jeweils 652×19 Bit, die zusammen den Mikroprogrammspeicher darstellen.

`bcdcalc` enthält 42 Padzellen: 12 Eingangs-, 24 Ausgangs- und 6 Spannungsversorgungspads. Da der Entwurf von der Fläche Core-bestimmt ist, wurden an zwei Seiten des Chips

„liegende“ Padzellen benutzt, um die Gesamtfläche zu verringern. SOLO 1400 erlaubt, mit einigen Einschränkungen, solch ein „Mischen“ der Padzell-Bibliotheken.

Die maximale Taktfrequenz der Schaltung wird durch die Zugriffe auf die (relativ) langsamen Speicher begrenzt. Bei den von uns durchgeführten Simulationen der Schaltung (worst case, größte Verzögerungszeiten) funktionierte die Schaltung, bei gleichlangen Clockphasen, bis zu einer Taktfrequenz von circa 4 MHz fehlerfrei. Da durch die Trennung der Taktleitungen in `clock` und `romclock` das Timing noch genauer auf die Verzögerungszeiten der Schaltung abgestimmt werden kann, ist eine Arbeitsfrequenz über 5 MHz zu erwarten.

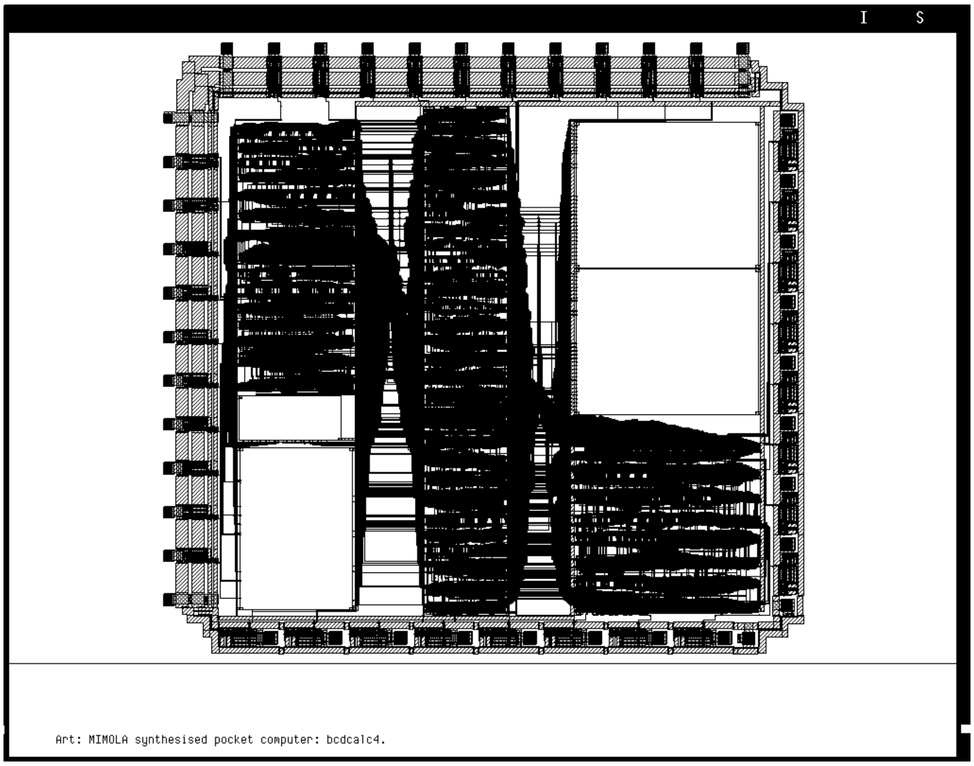


Abbildung 5: SOLO 1400 Layout für `bcdcalc`

4.4 Ergebnis eines **CADENCE/SOLO 2030**-Layouts

Mit den zu MIM2SOLO gehörenden Werkzeugen ist auch die Umsetzung der MIMOLA/MODEL Beschreibung in andere Netzlistenformate möglich. Vor allem um die entsprechenden Programme zu testen, wurde die `bcdcalc` Beschreibung von MODEL in ein CADENCE/SOLO 2030 Schematic umgesetzt und mit CADENCE/SOLO 2030 ein Chiplayout für `bcdcalc` erstellt.

Da sowohl SOLO 1400 als auch CADENCE/SOLO 2030 auf dem selben Herstellungsprozeß (ES2 ecpd15) beruhen, können damit dann auch die Layoutstrategien und die Layoutqualität der Entwurfswerkzeuge an einem wirklichen Entwurf verglichen werden. Allerdings verwendet CADENCE/SOLO 2030 wirklich Standardzellen, während SOLO 1400 eine Art Gatematrix-Layout Strategie implementiert; aufgrund des Prozesses können jedoch vergleichbare Größen der Gatter angenommen werden.

Die Erfahrungen mit dem verwendeten MIM2SOLO Programm `idl2skill` sind dabei durchweg positiv. Zunächst wird für die Umsetzung die MODEL Beschreibung neu übersetzt, mit Referenzen zu einer RT-Bibliothek `rtlib-2030`, die die für CADENCE/SOLO 2030 zur Verfügung stehenden Standardzellen beschreibt. Auch die Umsetzung der im Entwurf verwendeten Makrozellen (RAM, ROM, etc.) ist möglich, da SOLO 1400 und CADENCE/SOLO 2030 für ecpd15 die selben Generatoren verwenden. `idl2skill` liest dann die MODEL Netzliste ein und erzeugt mit einer kleinen SKILL Prozedur ein (flaches) CADENCE/SOLO 2030 Schematic. Nur die Padzellen müssen anschließend noch manuell mit den Netznamen für die globalen Ein- und Ausgänge versehen werden. Danach kann das Schematic wie üblich für die Simulation und die Layouterstellung genutzt werden.

Das mit CADENCE/SOLO 2030 erstellte Chiplayout ist in Abbildung 6 dargestellt. Da das CADENCE/SOLO 2030 System keinen automatischen Floorplanner enthält, mußten die Makrozellen (RAM, ROM) und die für Standardzellen verwendeten Bereiche manuell plaziert werden. Der Bedienungsaufwand zum Erstellen eines Layouts ist damit wesentlich höher als beim SOLO 1400 System. Durch die bessere Plazierung und globale Verdrahtung, die auf Heuristiken und Simulated Annealing beruht, ist der Core des Chips mit etwa 20 mm^2 deutlich kompakter als beim SOLO 1400 System. Dies ist auch in der Abbildung leicht zu erkennen. Die für das gesamte Chip inklusive I/O Pads benötigte Fläche beträgt 36 mm^2.

Allerdings ergaben sich nach dem Layout für etwa 10 Netze Fanout-Verletzungen durch zu hohe Leitungskapazitäten, da die Treiberleistungen einiger ES2 Standardzellen sehr niedrig liegen. Die Möglichkeit des CADENCE/SOLO 2030 Systems, einige Netze mit höherer Priorität bei der Plazierung und Globalverdrahtung zu berücksichtigen, brachte dabei bei einem zweiten Versuch keine Verbesserung.

Da zu diesem Zeitpunkt das SOLO 1400 Layout bereits fertiggestellt war, wurde auf die

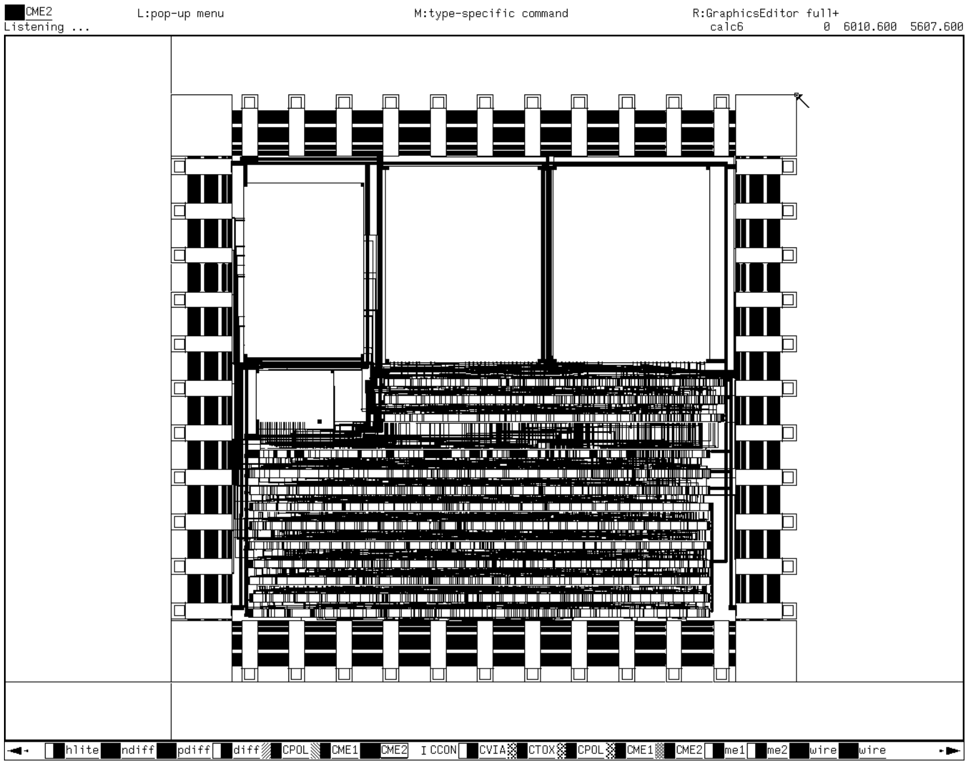


Abbildung 6: CADENCE/SOLO 2030 Layout für `bcdcalc`

weitere Arbeit — den nötigen Einbau von Treibern für die kritischen Netze — zugunsten des SOLO 1400 Entwurfs verzichtet. Der Flächenvorteil des CADENCE/SOLO 2030 Layouts wäre durch die zusätzlichen Treiber verringert worden, und die Einsparung von etwa 5 mm^2 hätte für die geringen zu fertigenden Stückzahlen den Aufwand nicht gerechtfertigt.

Für größere und nicht wie `bcdcalc` durch Makrozellen dominierte Entwürfe dürfte die Flächendifferenz zwischen SOLO 1400 und CADENCE/SOLO 2030 Layouts aber noch wesentlich deutlicher ausfallen und damit die Vorteile der CADENCE/SOLO 2030 Layoutalgorithmen aufzeigen.

A Technische Referenz für bcdcalc

A.1 I/O Spezifikation für bcdcalc

A.1.1 Eingänge

`reset` (active high)

asynchroner Reset des Rechners. Für eine sichere Funktion muß `reset` mindestens während zwei Taktphasen aktiv sein

`clock` Takteingang. Auf der steigenden Flanke von `clock` werden alle internen Register (auch der μPC) geladen, die fallende Flanke dient zur Generierung des WE (write-enable) Signals für die RAM-Makrozellen

`romclock` Takteingang. Auf der steigenden Flanke von `romclock` wird das μROM aktiviert, mit der fallenden Flanke von romclock wird das ME (memory-enable = address-latch) Signal für die RAM-Makrozellen aktiv:

Daraus ergibt sich folgendes Taktschema für `clock` und `romclock`:

```
   Phase:     0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3
   reset:     .1111111111111111......................
   clock:     ..1111....1111....1111....1111....1111..
romclock:     ....1111....1111....1111....1111....1111
```

`FctKeyIn` (active high)

Eingabe einer Funktion. Während bcdcalc Rechenergebnisse ausgibt (Prozedur 'Anzeige'), wird der Wert von `FctKeyIn` und `NumKeyIn` überwacht. Sobald die Leitung `FctKeyIn` den Wert Eins annimmt, wird `RZeichen` mit dem aktuellen Wert von `Ri0` (welches das aktive Digit selektiert) geladen und als Tastendruck gespeichert. Für `FctKeyIn` wird außerdem `RNoFunction` auf Null (d.h. „Funktion“) gesetzt.

Wird anschließend während eines kompletten Durchlaufs über alle 16 Digitleitungen kein Tastendruck festgestellt (d.h. `FctKeyIn` und `NumKeyIn` beide Null), wird der letzte in `RZeichen` gespeicherte Wert als aktuelle Eingabe interpretiert.

`NumKeyIn` (active high)

Eingabe einer Ziffer. Während bcdcalc Rechenergebnisse ausgibt (Prozedur 'Anzeige'), wird der Wert von `FctKeyIn` und `NumKeyIn` überwacht. Sobald die Leitung `NumKeyIn` den Wert Eins annimmt, wird `RZeichen` mit dem aktuellen Wert von `Ri0` (welches das aktive Digit selektiert) geladen und als Tastendruck gespeichert. Für `FctKeyIn` wird außerdem `RNoFunction` auf Eins (d.h. „Ziffer“) gesetzt.

Wird anschließend während eines kompletten Durchlaufs über alle 16 Digitleitungen kein Tastendruck festgestellt (d.h. `FctKeyIn` und `NumKeyIn` beide Null), wird der letzte in `RZeichen` gespeicherte Wert als aktuelle Eingabe interpretiert.

`testin` (active high)

Selektiert den Testmodus. Im normalen Betrieb ist die Leitung konstant Null.

Wenn `testin` auf Eins gesetzt ist, wenn `bcdcalc` nach Abschluß einer Rechenoperation die Prozedur 'Anzeige' beginnt, wartet `bcdcalc` in einer Schleife darauf, daß `testin` wieder Null wird. Anschließend liest `bcdcalc` die Werte vom Bus `inport(0:4)` und gibt dann den aktuellen Inhalt des X-Registers einmal über `calcout(0:23)` aus. Die Werte von `inport(0:3)` werden als Eingabezeichen interpretiert (d.h. in `RZeichen` geladen), der Wert von `inport(4)` gibt an, ob eine Funktion (`inport(4)` Null) oder eine Ziffer (`inport(4)` Eins) eingegeben werden soll.

Um bcdcalc zuverlässig im Testmode zu betreiben, sollte `testin` auf Eins gelegt und dann ein Reset ausgelöst werden. Anschließend sollte dann testin für jede Eingabe (ein Sample von `inport(0:4)`) jeweils für 16 Zyklen (mindestens vier) auf Null gesetzt werden.

`inport(0:4)` (active high)

Eingabebus für den Testmodus. Im normalen Betrieb wird `inport(0:4)` konstant auf Null gelegt.

Der Wert von `inport(0:4)` wird im Testmode, ein bis drei Zyklen nachdem `testin` den Wert Null annimmt, eingelesen. `inport(4)` selektiert Funktion (Null) oder Ziffer (Eins), `inport(0:3)` spezifizieren den Wert.

Im Scanmodus haben die Eingänge `inport(0:4)` eine besondere Bedeutung, siehe die Beschreibung von `scanact`.

`scanact` (active high)

Selektiert den Scanmodus. Im Scanmodus werden die `inport` und `calcout` als Ein- und Ausgänge der Schaltung benutzt.

Dabei wird zum einen ein Scanpath durch den Mikroprogrammzähler aktiviert, der es erlaubt, gezielt Routinen der Arithmetik von `bcdcalc` anzuspringen und zu testen. Dazu werden folgende Leitungen benutzt: `inport(0)` dient als Scanclock, `inport(1)` ist der Eingang des Scanpfads und `calcout(19)` dessen Ausgang.

Zum anderen kann der Inhalt der beiden Mikroprogramm-ROMs an den Ausgängen der Schaltung beobachtet werden. Zusammen mit der Ladbarkeit des Mikroprogrammzählers über den Scaneingang kann so ein kompletter Test des Mikroprogramms durchgeführt werden. Die Leitung `inport(2)` dient dabei als Selektionseingang: wobei im Fall `inport(2) = 0` die Werte des ersten ROMs an `calcout(0:18)` anliegen, im anderen Fall die Werte des zweiten Mikroprogrammspeichers.

A.1.2 Ausgänge

`calcout(0:23)` (active high)

Bus zur Ansteuerung einer gemultiplexten Sieben-Segment Anzeige.

Im Scanmodus dient `calcout(0:23)` außerdem zum Auslesen des μROMs und als Ausgang des Scanpfads. Siehe die Beschreibung von `scanact`.

Die Bits `calcout(0:15)` sind die 16 Digit-Ausgänge von `bcdcalc`. Jeweils eine dieser Leitungen ist aktiv (Eins), alle anderen sind inaktiv.

Die Ziffern der Mantisse werden von `calcout(10)` (MSB, vor dem Dezimalpunkt) bis `calcout(0)` (LSB) angesteuert. Der Exponent wird von `calcout(12)` (MSB) und `calcout(11)` (LSB) angesteuert, das Vorzeichen der Mantisse von `calcout(13)`, und `calcout(14)` ist das Vorzeichen des Exponenten. Der Ausgang `calcout(15)` steuert keine Ziffern, wird aber für die Eingabe von Funktionen benötigt, siehe die Beschreibung von `FctKeyIn` und `NumKeyIn`.

Die Bits `calcout(16:23)` sind die Segment-Ausgänge zur direkten Ansteuerung von Sieben-Segment Anzeigen. Bei einer Anordnung der Segmente gemäß

```
 --A--
B     C
 --D--
E     F
 --G--
        .DP
```

gilt `calcout(23)` = A, `calcout(22)` = B, `calcout(21)` = C, `calcout(20)` = D, `calcout(19)` = E, `calcout(18)` = F, `calcout(17)` = G, `calcout(16)` = DP.

Da alle Leitungen active high sind, wird zum Anschluß der Segmente noch ein Satz invertierender Treiber für die Digitleitungen benötigt.

A.2 Tastencodes

Die folgende Tabelle 1 gibt die zu dem jeweiligen Wert i gehörende Funktion von `FctKeyIn` und `NumKeyIn` an. Die entsprechende Funktionen wird also ausgelöst, wenn `FctKeyIn` bzw. `NumKeyIn` auf Eins liegen, während die Digitleitung i aktiv ist.

Im Testmode müssen die entsprechenden Werte an `inport(0:4)` angelegt werden. Dabei selektiert `inport(4)` zwischen Funktion (`inport(4)` Null) und den Ziffernbefehlen (`inport(4)` Eins). Die vier Bits `inport(3:0)` werden auf den Wert i gesetzt.

Code	`FctKeyIn`	`NumKeyIn`
0000	Nop	Ziffer Null
0001	= (Auswertung aller Op.)	Ziffer Eins
0010	Addition	Ziffer Zwei
0011	Subtraktion	Ziffer Drei
0100	Multiplikation	Ziffer Vier
0101	Division	Ziffer Fünf
0110	Nop	Ziffer Sechs
0111	Speicher Addition	Ziffer Sieben
1000	Speicher Multiplikation	Ziffer Acht
1001	Speicher Recall	Ziffer Neun
1010	Speicher Löschen	Exponent (Eingabe des Exp. starten)
1011	Nop	Vorzeichenwechsel (Mant. oder Exp.)
1100	Nop	CE (letzte Eingabe löschen)
1101	Nop	CLR (alle Eingaben löschen)
1110	Nop	Klammer auf
1111	Nop	Klammer zu

Tabelle 1: Funktionscodes für `bcdcalc`

A.3 Applikationsschaltung

Die folgende Abbildung 7 zeigt die Beschaltung von `bcdcalc` zu einem kompletten Taschenrechner. Dazu werden außer dem `bcdcalc`-Chip, den Sieben-Segment Anzeigen und den Eingabetasten nur noch ein Zwei-Phasen Taktgenerator und invertierende Treiber für die Digitleitungen benötigt.

Die Anordnung der einzelnen Sieben-Segment Anzeigen sollte natürlich nicht numerisch sortiert ($0 \ldots 13$) erfolgen, sondern in der Reihenfolge Vorzeichen Mantisse (`calcout(13)`), Mantisse MSB...LSB (`calcout(10:0)`), Vorzeichen Exponent (`calcout(14)`), Exponent (MSB,LSB) (`calcout(12:11)`).

Als Tasten können einfache einpolige Schalter/Taster verwendet werden, die eventuell notwendige Entprellung wird vom `bcdcalc`-Chip vorgenommen. Bei der Verwendung von einpoligen Tastern müssen natürlich die Digit-Ausgangsleitungen von `bcdcalc` mit Dioden oder Widerständen (etwa 10KΩ) gesichert werden, um Kurzschlüssen bei gleichzeitigem Drücken mehrerer Tasten vorzubeugen — `bcdcalc` verwendet keine Open-Collector Aus-

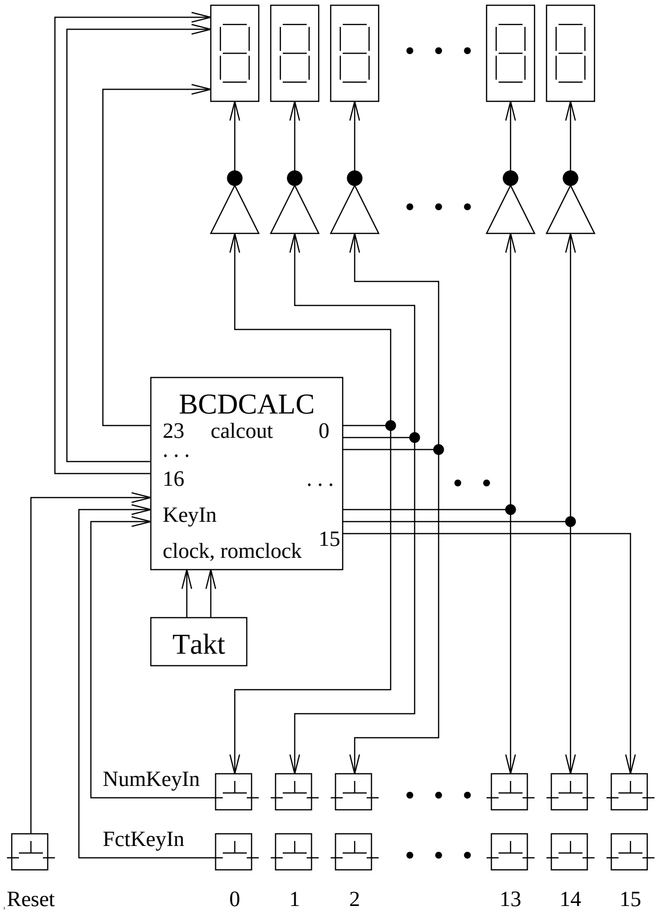


Abbildung 7: Beispielschaltung mit `bcdcalc`

gangstreiber.

Ebenfalls nicht in der Abbildung gezeigt sind die Abschlußwiderstände nach Masse für die `reset`, `NumKeyIn` und `FctKeyIn` Leitungen, um diese auf definierte Pegel zu ziehen, wenn keine Taste gedrückt wird.

Die nicht benötigten Eingabeleitungen `testin`, `scanact` und `inport(0:4)` werden konstant auf Masse gelegt.

B Dateien:

B.1 Die MIMOLA Beschreibung mim.bcdcalc

B.2 Die MODEL Beschreibung bcdcalc.mod

B.3 Simulationsstimuli: bcdcalc.wdl, bcdcalc.wdl_scan

B.4 Der Bondplan: bcdcalc.con

Pad to Pin table: – starting at the left edge, anticlockwise

Position		Pad	I/O	**DIL 48**	CC 68
LEFT	↓	gndpy1ins	GND	43	63
		clockins	input	44	64
		romclockins	input	45	65
		calcout1ins(0)	output	46	66
		calcout1ins(1)	output	47	67
		calcout1ins(2)	output	48	68
		calcout1ins(3)	output	1	1
		calcout1ins(4)	output	2	2
		calcout1ins(5)	output	3	3
		calcout1ins(5)	output	4	4
		calcout1ins(7)	output	5	5
		pwrcoins	VDD	6	6
BOTTOM	→	calcout2ins(0)	output	10	14
		calcout2ins(1)	output	11	15
		calcout2ins(2)	output	12	17
		calcout2ins(3)	output	13	18
		calcout2ins(4)	output	14	20
		calcout2ins(5)	output	15	21
		calcout2ins(6)	output	16	23
		calcout2ins(7)	output	17	24
		gndpy2ins	GND	18	25
RIGHT	↑	pwrpy2ins	VDD	20	29
		calcout2ins(8)	output	21	31
		calcout2ins(9)	output	22	32
		calcout2ins(10)	output	23	33
		calcout2ins(11)	output	24	35
		calcout2ins(12)	output	25	36
		calcout2ins(13)	output	26	37
		calcout2ins(14)	output	27	39
		calcout2ins(15)	output	28	40
TOP	←	gndcoins	GND	31	46
		scanactins	input	32	47
		testinins	input	33	48
		inportins(4)	input	34	49
		inportins(3)	input	35	50
		inportins(2)	input	36	51
		inportins(1)	input	37	53
		inportins(0)	input	38	54
		fctkeyinins	input	39	55
		numkeyinins	input	40	56
		resetins	input	41	57
		pwrpy1ins	VDD	42	58

Literatur

[ES2 90] European Silicon Structures Ltd., *SOLO 1400 Release 3.0 Reference Manual*, Berkshire, 1990

[ES2 91] European Silicon Structures Ltd., *SOLO 1400 Release 3.1 Reference Manual*, Berkshire, 1991

[Hendrich, Lohse & Rauscher 92a] N. Hendrich, J. Lohse & R. Rauscher, *mim2solo: Automatischer Entwurf mikroprogrammierter VLSI Schaltungen mit MIMOLA und SOLO 1400*, Bericht 201/92, Fachbereich Informatik, Universität Hamburg, 1992

[Hendrich, Lohse & Rauscher 92b] N. Hendrich, J. Lohse & R. Rauscher, *Silicon Compilation and Rapid Prototyping of Microprogrammed VLSI-Circuits with MIMOLA and SOLO 1400*, Proc. EUROMICRO-92, 287–294, 1992

[Jöhnk & Marwedel 89] R. Jöhnk & P. Marwedel, *Mimola Reference Manual V.3.45*, interner Bericht 8902, Christian-Albrechts-Universität Kiel, 1989

[Ku & Micheli 90] D. Ku & G. De Micheli, *High Level Synthesis and Optimization Strategies in Hercules and Hebe*, Proc. EuroASIC 1990, 124–129, 1990

[Mäder 93] A. Mäder, *minrom: Ein effizienter Algorithmus zur Kompaktierung von horizontalen Mikroprogrammen*, private communication, 1993

[Mäder, Hendrich & Lagemann 93] A. Mäder, N. Hendrich & K. Lagemann, *Teaching High-Level Design*, Proc. Eurochip Workshop 93, Toledo, 1993, to appear

[Marwedel 85] P. Marwedel, *Ein Software-System zur Synthese von Rechnerstrukturen und zur Erzeugung von Mikrocode*, Habilitation, Christian-Albrechts-Universität Kiel, 1985